Polyhedral Compilation without Polyhedra

Sven Verdoolaege

INRIA and KU Leuven
Sven.Verdoolaege@inria.fr

January 20, 2015

Outline

Part I

Polyhedral Compilation

Outline

Polyhedral Compilation

Polyhedral Compilation

Analyzing and/or transforming loop programs using the *polyhedral model*

Polyhedral Model

Abstract representation of a loop program

- instance based
  - ⇒ statement *instances*
  - ⇒ array *elements*
- compact representation based on polyhedra or similar objects
  - ⇒ integer points in unions of parametric polyhedra
  - ⇒ Presburger sets and relations
- parametric
  - ⇒ description may depend on symbolic constants

[8, 10, 14, 16]

Polyhedral Compilation

Polyhedral Compilation

Analyzing and/or transforming loop programs using the *polyhedral model*

Polyhedral Model

Abstract representation of a loop program

- instance based
  - ⇒ statement *instances*
  - ⇒ array *elements*
- compact representation based on polyhedra or similar objects
  - ⇒ integer points in unions of parametric polyhedra
  - ⇒ Presburger sets and relations
- parametric
  - ⇒ description may depend on symbolic constants

Note: naming is "historical"

- polyhedral compilation does not require polyhedra (e.g., `omega(+)`)
- other approaches also use polyhedra (e.g., abstract interpretation)

Polyhedral Model

Main constituents of program representation

- Instance Set
  - $\Rightarrow$ the set of all statement instances
- Access Relations
  - $\Rightarrow$ the array elements accessed by a statement instance
- Dependences
  - $\Rightarrow$ the statement instances that depend on a statement instance
- Schedule
  - $\Rightarrow$ the relative execution order of statement instances

Polyhedral Model Requirements

Requirements for basic polyhedral model: SANA input

- Static control
  $\Rightarrow$ control does not depend on input data
- Affine
  $\Rightarrow$ all relevant expressions are (quasi-)affine
- No Aliasing
  $\Rightarrow$ essentially no pointer manipulations

Polyhedral Model Requirements

Requirements for basic polyhedral model: SANA input

- Static control
  $\Rightarrow$ control does not depend on input data
- Affine
  $\Rightarrow$ all relevant expressions are (quasi-)affine
- No Aliasing
  $\Rightarrow$ essentially no pointer manipulations

Note:

- polyhedral model may be *approximation* of input that does not strictly satisfy all requirements
- many *extensions* are available
  a small selection of these extensions will be discussed in this tutorial

Illustrative Example

```
R:   h(A[2]);
     for (int i = 0; i < 2; ++i)
          for (int j = 0; j < 2; ++j)
S:             A[i + j] = f(i, j);
     for (int k = 0; k < 2; ++k)
T:             g(A[k], A[0]);
```

Illustrative Example

```
R:   h(A[2]);
     for (int i = 0; i < 2; ++i)
          for (int j = 0; j < 2; ++j)
S:             A[i + j] = f(i, j);
     for (int k = 0; k < 2; ++k)
T:             g(A[k], A[0]);
```

- instance based

- Instance Set (set of statement instances)

$$I = \{\mathtt{R}(); \mathtt{S}(0,0); \mathtt{S}(0,1); \mathtt{S}(1,0); \mathtt{S}(1,1); \mathtt{T}(0); \mathtt{T}(1)\}$$

Illustrative Example

```
R:   h(A[2]);
     for (int i = 0; i < 2; ++i)
         for (int j = 0; j < 2; ++j)
S:           A[i + j] = f(i, j);
     for (int k = 0; k < 2; ++k)
T:           g(A[k], A[0]);
```

- instance based

- Instance Set (set of statement instances)

$$I = \{ \mathtt{R}(); \mathtt{S}(0,0); \mathtt{S}(0,1); \mathtt{S}(1,0); \mathtt{S}(1,1); \mathtt{T}(0); \mathtt{T}(1) \}$$

- Access Relations (accessed array elements; W: write, R: read)

$$W = \{ \mathtt{S}(0,0) \to \mathtt{A}(0); \mathtt{S}(0,1) \to \mathtt{A}(1); \mathtt{S}(1,0) \to \mathtt{A}(1); \mathtt{S}(1,1) \to \mathtt{A}(2) \}$$

$$R = \{ \mathtt{R}() \to \mathtt{A}(2); \mathtt{T}(0) \to \mathtt{A}(0); \mathtt{T}(1) \to \mathtt{A}(1); \mathtt{T}(1) \to \mathtt{A}(0) \}$$

Illustrative Example

```
R:   h(A[2]);
     for (int i = 0; i < 2; ++i)
         for (int j = 0; j < 2; ++j)
S:           A[i + j] = f(i, j);
     for (int k = 0; k < 2; ++k)
T:           g(A[k], A[0]);
```

- instance based

- Instance Set (set of statement instances)

$$I = \{ \mathtt{R}(); \mathtt{S}(0,0); \mathtt{S}(0,1); \mathtt{S}(1,0); \mathtt{S}(1,1); \mathtt{T}(0); \mathtt{T}(1) \}$$

- Access Relations (accessed array elements; W: write, R: read)

$$W = \{ \mathtt{S}(0,0) \to \mathtt{A}(0); \mathtt{S}(0,1) \to \mathtt{A}(1); \mathtt{S}(1,0) \to \mathtt{A}(1); \mathtt{S}(1,1) \to \mathtt{A}(2) \}$$

$$R = \{ \mathtt{R}() \to \mathtt{A}(2); \mathtt{T}(0) \to \mathtt{A}(0); \mathtt{T}(1) \to \mathtt{A}(1); \mathtt{T}(1) \to \mathtt{A}(0) \}$$

- Schedule (relative execution order)

$$\mathtt{R}(), \mathtt{S}(0,0), \mathtt{S}(0,1), \mathtt{S}(1,0), \mathtt{S}(1,1), \mathtt{T}(0), \mathtt{T}(1)$$

Illustrative Example

```
R:   h(A[2]);
     for (int i = 0; i < 2; ++i)
         for (int j = 0; j < 2; ++j)
S:           A[i + j] = f(i, j);
     for (int k = 0; k < 2; ++k)
T:           g(A[k], A[0]);
```

- instance based
- compact representation

- Instance Set (set of statement instances)

$$I = \{ \mathtt{R}(); \mathtt{S}(0,0); \mathtt{S}(0,1); \mathtt{S}(1,0); \mathtt{S}(1,1); \mathtt{T}(0); \mathtt{T}(1) \}$$

- Access Relations (accessed array elements: Writes W, Reads R)

$$\{ \mathtt{R}(); \mathtt{S}(i,j) : 0 \leq i < 2 \wedge 0 \leq j < 2; \mathtt{T}(k) : 0 \leq k < 2; \}$$

$$W = \{ \mathtt{S}(0,0) \to \mathtt{A}(0); \mathtt{S}(0,1) \to \mathtt{A}(1); \mathtt{S}(1,0) \to \mathtt{A}(1); \mathtt{S}(1,1) \to \mathtt{A}(2) \}$$

$$R = \{ \mathtt{R}() \to \mathtt{A}(2); \mathtt{T}(0) \to \mathtt{A}(0); \mathtt{T}(1) \to \mathtt{A}(1); \mathtt{T}(1) \to \mathtt{A}(0) \}$$

- Schedule (relative execution order)

$$\mathtt{R}(), \mathtt{S}(0,0), \mathtt{S}(0,1), \mathtt{S}(1,0), \mathtt{S}(1,1), \mathtt{T}(0), \mathtt{T}(1)$$

Illustrative Example

```
R:   h(A[2]);
     for (int i = 0; i < 2; ++i)
          for (int j = 0; j < 2; ++j)
S:             A[i + j] = f(i, j);
     for (int k = 0; k < 2; ++k)
T:             g(A[k], A[0]);
```

- instance based
- compact representation

- Instance Set (set of statement instances)

$$I = \{ \mathtt{R}(); \mathtt{S}(0,0); \mathtt{S}(0,1); \mathtt{S}(1,0); \mathtt{S}(1,1); \mathtt{T}(0); \mathtt{T}(1) \}$$

- Access Relations (accessed array elements: Memory Read/Write)

$$\{ \mathtt{R}(); \mathtt{S}(i,j) : 0 \le i < 2 \wedge 0 \le j < 2; \mathtt{T}(k) : 0 \le k < 2; \}$$

$W = \{ \mathtt{S}(0,0) \to \mathtt{A}(0); \mathtt{S}(0,1) \to \mathtt{A}(1); \mathtt{S}(1,0) \to \mathtt{A}(1); \mathtt{S}(1,1) \to \mathtt{A}(2) \}$

$R = \{ \mathtt{R}() \to \mathtt{A}(2); \mathtt{T}(0) \to \mathtt{A}(0); \mathtt{T}(1) \to \mathtt{A}(1); \mathtt{T}(1) \to \mathtt{A}(0) \}$

Constraints

- "read off", or
- obtained through abstract interpretation

- Schedule (relative execution order)

$$\mathtt{R}(), \mathtt{S}(0,0), \mathtt{S}(0,1), \mathtt{S}(1,0), \mathtt{S}(1,1), \mathtt{T}(0), \mathtt{T}(1)$$

Illustrative Example

```
R:   h(A[2]);
     for (int i = 0; i < 2; ++i)
          for (int j = 0; j < 2; ++j)
S:             A[i + j] = f(i, j);
     for (int k = 0; k < 2; ++k)
T:             g(A[k], A[0]);
```

- instance based
- compact representation

- Instance Set (set of statement instances)

$$I = \{ \mathtt{R}(); \mathtt{S}(i,j) : 0 \leq i < 2 \wedge 0 \leq j < 2; \mathtt{T}(k) : 0 \leq k < 2 \}$$

- Access Relations (accessed array elements; W: write, R: read)

$$W = \{ \mathtt{S}(0,0) \rightarrow \mathtt{A}(0); \mathtt{S}(0,1) \rightarrow \mathtt{A}(1); \mathtt{S}(1,0) \rightarrow \mathtt{A}(1); \mathtt{S}(1,1) \rightarrow \mathtt{A}(2) \}$$

$$R = \{ \mathtt{R}() \rightarrow \mathtt{A}(2); \mathtt{T}(0) \rightarrow \mathtt{A}(0); \mathtt{T}(1) \rightarrow \mathtt{A}(1); \mathtt{T}(1) \rightarrow \mathtt{A}(0) \}$$

- Schedule (relative execution order)

$$\mathtt{R}(), \mathtt{S}(0,0), \mathtt{S}(0,1), \mathtt{S}(1,0), \mathtt{S}(1,1), \mathtt{T}(0), \mathtt{T}(1)$$

Illustrative Example

```
R:   h(A[2]);
     for (int i = 0; i < 2; ++i)
          for (int j = 0; j < 2; ++j)
S:             A[i + j] = f(i, j);
     for (int k = 0; k < 2; ++k)
T:             g(A[k], A[0]);
```

- instance based
- compact representation

- Instance Set (set of statement instances)

$$I = \{\mathtt{R}(); \mathtt{S}(i,j) : 0 \le i < 2 \wedge 0 \le j < 2; \mathtt{T}(k) : 0 \le k < 2\}$$

- Access Relations (accessed array elements; W: write, R: read)

$$W = \{\mathtt{S}(0,0) \to \mathtt{A}(0); \mathtt{S}(0,1) \to \mathtt{A}(1); \mathtt{S}(1,0) \to \mathtt{A}(1); \mathtt{S}(1,1) \to \mathtt{A}(2)\}$$

$$R = \{\{\mathtt{S}(i,j) \to \mathtt{A}(i+j) : 0 \le i < 2 \wedge 0 \le j < 2\} \to \mathtt{A}(0)\}$$

- Schedule (relative execution order)

$$\mathtt{R}(), \mathtt{S}(0,0), \mathtt{S}(0,1), \mathtt{S}(1,0), \mathtt{S}(1,1), \mathtt{T}(0), \mathtt{T}(1)$$

Illustrative Example

```
R:   h(A[2]);
     for (int i = 0; i < 2; ++i)
          for (int j = 0; j < 2; ++j)
S:             A[i + j] = f(i, j);
     for (int k = 0; k < 2; ++k)
T:             g(A[k], A[0]);
```

- instance based
- compact representation

- Instance Set (set of statement instances)

$$I = \{ \mathtt{R}(); \mathtt{S}(i,j) : 0 \leq i < 2 \wedge 0 \leq j < 2; \mathtt{T}(k) : 0 \leq k < 2 \}$$

- Access Relations (accessed array elements; W: write, R: read)

$$W = \{ \mathtt{S}(i,j) \rightarrow \mathtt{A}(i+j) : 0 \leq i < 2 \wedge 0 \leq j < 2 \}$$

$$R = \{ \mathtt{R}() \rightarrow \mathtt{A}(2); \mathtt{T}(0) \rightarrow \mathtt{A}(0); \mathtt{T}(1) \rightarrow \mathtt{A}(1); \mathtt{T}(1) \rightarrow \mathtt{A}(0) \}$$

- Schedule (relative execution order)

$$\mathtt{R}(), \mathtt{S}(0,0), \mathtt{S}(0,1), \mathtt{S}(1,0), \mathtt{S}(1,1), \mathtt{T}(0), \mathtt{T}(1)$$

Illustrative Example

```
R:   h(A[2]);
     for (int i = 0; i < 2; ++i)
         for (int j = 0; j < 2; ++j)
S:           A[i + j] = f(i, j);
     for (int k = 0; k < 2; ++k)
T:           g(A[k], A[0]);
```

- instance based
- compact representation

- Instance Set (set of statement instances)

$$I = \{\mathtt{R}(); \mathtt{S}(i,j) : 0 \le i < 2 \wedge 0 \le j < 2; \mathtt{T}(k) : 0 \le k < 2\}$$

- Access Relations (accessed array elements; W: write, R: read)

$$W = \{\mathtt{S}(i,j) \to \mathtt{A}(i+j) : 0 \le i < 2 \wedge 0 \le j < 2\}$$

$$R = \{\mathtt{R}() \to \mathtt{A}(2); \mathtt{T}(0) \to \mathtt{A}(0); \mathtt{T}(1) \to \mathtt{A}(1); \mathtt{T}(1) \to \mathtt{A}(0)\}$$

- Schedule (relative execution order)

$$\{\mathtt{R}() \to \mathtt{A}(2); \mathtt{T}(k) \to \mathtt{A}(0) : 0 \le k < 2; \mathtt{T}(k) \to \mathtt{A}(k) : 0 \le k < 2\}$$

$$\mathtt{R}(), \mathtt{S}(0,0), \mathtt{S}(0,1), \mathtt{S}(1,0), \mathtt{S}(1,1), \mathtt{T}(0), \mathtt{T}(1)$$

Illustrative Example

```
R:   h(A[2]);
     for (int i = 0; i < 2; ++i)
          for (int j = 0; j < 2; ++j)
S:             A[i + j] = f(i, j);
     for (int k = 0; k < 2; ++k)
T:             g(A[k], A[0]);
```

- instance based
- compact representation

- Instance Set (set of statement instances)

$$I = \{\mathtt{R}(); \mathtt{S}(i,j) : 0 \leq i < 2 \wedge 0 \leq j < 2; \mathtt{T}(k) : 0 \leq k < 2\}$$

- Access Relations (accessed array elements; W: write, R: read)

$$W = \{\mathtt{S}(i,j) \rightarrow \mathtt{A}(i+j) : 0 \leq i < 2 \wedge 0 \leq j < 2\}$$

$$R = \{\mathtt{R}() \rightarrow \mathtt{A}(2); \mathtt{T}(k) \rightarrow \mathtt{A}(0) : 0 \leq k < 2; \mathtt{T}(k) \rightarrow \mathtt{A}(k) : 0 \leq k < 2\}$$

- Schedule (relative execution order)

$$\mathtt{R}(), \mathtt{S}(0,0), \mathtt{S}(0,1), \mathtt{S}(1,0), \mathtt{S}(1,1), \mathtt{T}(0), \mathtt{T}(1)$$

Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

- Instance Set (set of statement instances)

$$\{\, \mathtt{S1}(i,j) : 0 \le i < M \wedge 0 \le j < N;$$
$$\mathtt{S2}(i,j,k) : 0 \le i < M \wedge 0 \le j < N \wedge 0 \le k < K \,\}$$

- Access Relations (accessed array elements; W: write, R: read)

$$W = \{\, \mathtt{S1}(i,j) \to \mathtt{C}(i,j); \mathtt{S2}(i,j,k) \to \mathtt{C}(i,j) \,\}$$
$$R = \{\, \mathtt{S2}(i,j,k) \to \mathtt{C}(i,j); \mathtt{S2}(i,j,k) \to \mathtt{A}(i,k); \mathtt{S2}(i,j,k) \to \mathtt{B}(k,j) \,\}$$

- Schedule (relative execution order)

$$\mathtt{S1}(0,0), \mathtt{S2}(0,0,0), \mathtt{S2}(0,0,1), \ldots, \mathtt{S1}(0,1), \mathtt{S2}(0,1,0), \mathtt{S2}(0,1,1), \ldots,$$

Named Presburger Sets and Relations

[20]

Examples

$$\{\, \mathtt{R}(); \mathtt{S}(i,j) : 0 \le i < 2 \wedge 0 \le j < 2; \mathtt{T}(k) : 0 \le k < 2 \,\}$$
$$\{\, \mathtt{R}() \to \mathtt{A}(2); \mathtt{T}(k) \to \mathtt{A}(0) : 0 \le k < 2; \mathtt{T}(k) \to \mathtt{A}(k) : 0 \le k < 2 \,\}$$

General form

- Sets
$$\{\, S_1(\mathbf{i}) : f_1(\mathbf{i}); S_2(\mathbf{i}) : f_2(\mathbf{i}); \ldots \},$$
with f_k Presburger formulas
$\Rightarrow$ set of elements of the form $S_1(\mathbf{i})$, one for each $\mathbf{i}$ satisfying $f_1(\mathbf{i})$, ...
- Binary relations
$$\{\, S_1(\mathbf{i}) \to T_1(\mathbf{j}) : f_1(\mathbf{i},\mathbf{j}); S_2(\mathbf{i}) \to T_2(\mathbf{j}) : f_2(\mathbf{i},\mathbf{j}); \ldots \}$$
$\Rightarrow$ set of pairs of elements of the form $S_1(\mathbf{i}) \to T_1(\mathbf{j})$

Named Presburger Sets and Relations

[20]

Examples

$$\{\, \mathtt{R}(); \mathtt{S}(i,j) : 0 \le i < 2 \wedge 0 \le j < 2; \mathtt{T}(k) : 0 \le k < 2 \,\}$$

$$\{\, \mathtt{R}() \to \mathtt{A}(2); \mathtt{T}(k) \to \mathtt{A}(0) : 0 \le k < 2; \mathtt{T}(k) \to \mathtt{A}(k) : 0 \le k < 2 \,\}$$

General form

- Sets

$$\{\, S_1(\mathbf{i}) : f_1(\mathbf{i}); S_2(\mathbf{i}) : f_2(\mathbf{i}); \ldots \},$$

  with f_k Presburger formulas

  $\Rightarrow$ set of elements of the form $S_1(\mathbf{i})$, one for each $\mathbf{i}$ satisfying $f_1(\mathbf{i})$, ...

- Binary relations

$$\{\, S_1(\mathbf{i}) \to T_1(\mathbf{j}) : f_1(\mathbf{i},\mathbf{j}); S_2(\mathbf{i}) \to T_2(\mathbf{j}) : f_2(\mathbf{i},\mathbf{j}); \ldots \}$$

  $\Rightarrow$ set of pairs of elements of the form $S_1(\mathbf{i}) \to T_1(\mathbf{j})$

Note: despite "$\to$", not necessarily (single valued) functions

Named Presburger Sets and Relations

[20]

General form

- Sets

$$\{ S_1(\mathbf{i}) : f_1(\mathbf{i}); S_2(\mathbf{i}) : f_2(\mathbf{i}); \dots \},$$

where $f_k(\mathbf{i})$ are Presburger formulas with $\mathbf{i}$ as only free variables

$\Rightarrow$ set of elements of the form $S_1(\mathbf{i})$, one for each $\mathbf{i}$ satisfying $f_1(\mathbf{i})$, ...

Named Presburger Sets and Relations

[20]

General form

- Sets

$$\{ S_1(\mathbf{i}) : f_1(\mathbf{i}); S_2(\mathbf{i}) : f_2(\mathbf{i}); \dots \},$$

where $f_k(\mathbf{i})$ are Presburger formulas with $\mathbf{i}$ as only free variables

$\Rightarrow$ set of elements of the form $S_1(\mathbf{i})$, one for each $\mathbf{i}$ satisfying $f_1(\mathbf{i})$, ...

Note: may depend on interpretation of symbolic constants

$$\{ \mathrm{S}(i) : 0 \leq i \leq \mathrm{n}() \}$$

is equal to

$$\begin{cases} \emptyset & \text{if } \mathrm{n} < 0 \\ \{ \mathrm{S}(0) \} & \text{if } \mathrm{n} = 0 \\ \{ \mathrm{S}(0); \mathrm{S}(1) \} & \text{if } \mathrm{n} = 1 \\ \{ \mathrm{S}(0); \mathrm{S}(1); \mathrm{S}(2) \} & \text{if } \mathrm{n} = 2 \\ \dots \end{cases}$$

Quasi-Affine Expressions and Presburger Formulae

- Symbolic Constant
  - has unknown but fixed value
  - typically used to represent size parameter

Quasi-Affine Expressions and Presburger Formulae

- Symbolic Constant
  - has unknown but fixed value
  - typically used to represent size parameter
- Quasi-Affine Expression
  - variable
  - symbolic constant
  - integer constant
  - addition $(+)$, subtraction $(-)$
  - integer division by a constant $(\lfloor \cdot / d \rfloor)$

Quasi-Affine Expressions and Presburger Formulae

- Symbolic Constant
  - has unknown but fixed value
  - typically used to represent size parameter
- Quasi-Affine Expression $\leadsto$ Presburger Term
  - variable
  - symbolic constant
  - integer constant
  - addition $(+)$, subtraction $(-)$
  - integer division by a constant $(\lfloor \cdot / d \rfloor)$
- Presburger Formula
  - true
  - equality on terms $(=)$
  - less than or equal on terms $(\leq)$
  - logical connectives $(\wedge, \vee, \neg)$
  - quantification $(\exists, \forall)$

Syntactic Sugar

- false is equal to $\neg\mathrm{true}$

Syntactic Sugar

- false is equal to $\neg\mathrm{true}$
- $a \Rightarrow b$ is equal to $\neg a \vee b$

Syntactic Sugar

- false is equal to $\neg\mathrm{true}$
- $a \Rightarrow b$ is equal to $\neg a \vee b$
- $\{ S(\mathbf{i}) \}$ is equal to $\{ S(\mathbf{i}) : \mathrm{true} \}$

Syntactic Sugar

- false is equal to $\neg\mathrm{true}$
- $a \Rightarrow b$ is equal to $\neg a \vee b$
- $\{ S(\mathbf{i}) \}$ is equal to $\{ S(\mathbf{i}) : \mathrm{true} \}$
- $\{ S(i_1, \ldots i_{n-1}, g(i_1, \ldots, i_{n-1}), i_{n+1}, \ldots) : f(\mathbf{i}) \}$ is equal to $\{ S(i_1, \ldots i_{n-1}, i_n, i_{n+1}, \ldots) : f(\mathbf{i}) \wedge i_n = g(i_1, \ldots, i_{n-1}) \}$
  Example: $\{ \mathrm{S}(i) \rightarrow \mathrm{S}(i+1) \}$ is equal to $\{ \mathrm{S}(i) \rightarrow \mathrm{S}(j) : j = i+1 \}$

Syntactic Sugar

- false is equal to $\neg\mathrm{true}$
- $a \Rightarrow b$ is equal to $\neg a \vee b$
- $\{\, S(\mathbf{i}) \,\}$ is equal to $\{\, S(\mathbf{i}) : \mathrm{true} \,\}$
- $\{\, S(i_1, \ldots i_{n-1}, g(i_1, \ldots, i_{n-1}), i_{n+1}, \ldots) : f(\mathbf{i}) \,\}$ is equal to $\{\, S(i_1, \ldots i_{n-1}, i_n, i_{n+1}, \ldots) : f(\mathbf{i}) \wedge i_n = g(i_1, \ldots, i_{n-1}) \,\}$
  Example: $\{\, \mathrm{S}(i) \rightarrow \mathrm{S}(i+1) \,\}$ is equal to $\{\, \mathrm{S}(i) \rightarrow \mathrm{S}(j) : j = i+1 \,\}$
- n is equal to $n()$

Syntactic Sugar

- false is equal to $\neg\mathrm{true}$
- $a \Rightarrow b$ is equal to $\neg a \vee b$
- $\{\, S(\mathbf{i}) \,\}$ is equal to $\{\, S(\mathbf{i}) : \mathrm{true} \,\}$
- $\{\, S(i_1, \ldots i_{n-1}, g(i_1, \ldots, i_{n-1}), i_{n+1}, \ldots) : f(\mathbf{i}) \,\}$ is equal to $\{\, S(i_1, \ldots i_{n-1}, i_n, i_{n+1}, \ldots) : f(\mathbf{i}) \wedge i_n = g(i_1, \ldots, i_{n-1}) \,\}$
  Example: $\{\, \mathrm{S}(i) \rightarrow \mathrm{S}(i+1) \,\}$ is equal to $\{\, \mathrm{S}(i) \rightarrow \mathrm{S}(j) : j = i + 1 \,\}$
- n is equal to $n()$
- $a < b$ is equal to $a \leq b - 1$

Syntactic Sugar

- false is equal to $\neg\mathrm{true}$
- $a \Rightarrow b$ is equal to $\neg a \vee b$
- $\{ S(\mathbf{i}) \}$ is equal to $\{ S(\mathbf{i}) : \mathrm{true} \}$
- $\{ S(i_1, \ldots i_{n-1}, g(i_1, \ldots, i_{n-1}), i_{n+1}, \ldots) : f(\mathbf{i}) \}$ is equal to $\{ S(i_1, \ldots i_{n-1}, i_n, i_{n+1}, \ldots) : f(\mathbf{i}) \wedge i_n = g(i_1, \ldots, i_{n-1}) \}$
  Example: $\{ \mathrm{S}(i) \to \mathrm{S}(i+1) \}$ is equal to $\{ \mathrm{S}(i) \to \mathrm{S}(j) : j = i + 1 \}$
- n is equal to $n()$
- $a < b$ is equal to $a \leq b - 1$
- $a \geq b$ is equal to $b \leq a$

Syntactic Sugar

- false is equal to $\neg\mathrm{true}$
- $a \Rightarrow b$ is equal to $\neg a \vee b$
- $\{ S(\mathbf{i}) \}$ is equal to $\{ S(\mathbf{i}) : \mathrm{true} \}$
- $\{ S(i_1, \ldots i_{n-1}, g(i_1, \ldots, i_{n-1}), i_{n+1}, \ldots) : f(\mathbf{i}) \}$ is equal to $\{ S(i_1, \ldots i_{n-1}, i_n, i_{n+1}, \ldots) : f(\mathbf{i}) \wedge i_n = g(i_1, \ldots, i_{n-1}) \}$
  Example: $\{ \mathrm{S}(i) \rightarrow \mathrm{S}(i+1) \}$ is equal to $\{ \mathrm{S}(i) \rightarrow \mathrm{S}(j) : j = i+1 \}$
- n is equal to $n()$
- $a < b$ is equal to $a \leq b - 1$
- $a \geq b$ is equal to $b \leq a$
- $a > b$ is equal to $a \geq b + 1$

Syntactic Sugar

- false is equal to $\neg\mathrm{true}$
- $a \Rightarrow b$ is equal to $\neg a \vee b$
- $\{ S(\mathbf{i}) \}$ is equal to $\{ S(\mathbf{i}) : \mathrm{true} \}$
- $\{ S(i_1, \ldots i_{n-1}, g(i_1, \ldots, i_{n-1}), i_{n+1}, \ldots) : f(\mathbf{i}) \}$ is equal to $\{ S(i_1, \ldots i_{n-1}, i_n, i_{n+1}, \ldots) : f(\mathbf{i}) \wedge i_n = g(i_1, \ldots, i_{n-1}) \}$
  Example: $\{ \mathrm{S}(i) \rightarrow \mathrm{S}(i+1) \}$ is equal to $\{ \mathrm{S}(i) \rightarrow \mathrm{S}(j) : j = i+1 \}$
- n is equal to $n()$
- $a < b$ is equal to $a \leq b - 1$
- $a \geq b$ is equal to $b \leq a$
- $a > b$ is equal to $a \geq b + 1$
- $a, b \oplus c$ is equal to $a \oplus c \wedge b \oplus c$ with $\oplus \in \{ \leq, <, \geq, >, = \}$
  Example: $\{ \mathrm{S}(i,j) : i,j \geq 0 \}$ is equal to $\{ \mathrm{S}(i,j) : i \geq 0 \wedge j \geq 0 \}$

Syntactic Sugar

- false is equal to $\neg\mathrm{true}$
- $a \Rightarrow b$ is equal to $\neg a \vee b$
- $\{ S(\mathbf{i}) \}$ is equal to $\{ S(\mathbf{i}) : \mathrm{true} \}$
- $\{ S(i_1, \ldots i_{n-1}, g(i_1, \ldots, i_{n-1}), i_{n+1}, \ldots) : f(\mathbf{i}) \}$ is equal to $\{ S(i_1, \ldots i_{n-1}, i_n, i_{n+1}, \ldots) : f(\mathbf{i}) \wedge i_n = g(i_1, \ldots, i_{n-1}) \}$
  Example: $\{ \mathrm{S}(i) \to \mathrm{S}(i+1) \}$ is equal to $\{ \mathrm{S}(i) \to \mathrm{S}(j) : j = i+1 \}$
- n is equal to $n()$
- $a < b$ is equal to $a \leq b - 1$
- $a \geq b$ is equal to $b \leq a$
- $a > b$ is equal to $a \geq b + 1$
- $a, b \oplus c$ is equal to $a \oplus c \wedge b \oplus c$ with $\oplus \in \{ \leq, <, \geq, >, = \}$
  Example: $\{ \mathrm{S}(i,j) : i,j \geq 0 \}$ is equal to $\{ \mathrm{S}(i,j) : i \geq 0 \wedge j \geq 0 \}$
- $a \oplus_1 b \oplus_2 c$ is equal to $a \oplus_1 b \wedge b \oplus_2 c$ with $\{ \oplus_1, \oplus_2 \} \subset \{ \leq, <, \geq, >, = \}$
  Example: $\{ \mathrm{S}(i) : 0 \leq i \leq 10 \}$ is equal to $\{ \mathrm{S}(i) : 0 \leq i \wedge i \leq 10 \}$

Syntactic Sugar (2)

- $-e$ is equal to $0 - e$

Syntactic Sugar (2)

- $-e$ is equal to $0 - e$
- $n \cdot e$ is equal to $\underbrace{e + e + \cdots + e}_{n \text{ times}}$ (with n a non-negative integer constant)

Syntactic Sugar (2)

- $-e$ is equal to $0 - e$
- $n \cdot e$ is equal to $\underbrace{e + e + \cdots + e}_{n \text{ times}}$ (with n a non-negative integer constant)
- $a_1, \ldots, a_n \prec b_1, \ldots, b_n$ is equal to $\bigvee_{i=1}^{n} \left(\left(\bigwedge_{j=1}^{i-1} a_j = b_j \right) \wedge a_i < b_i \right)$

  Example: $\{ \mathrm{S}(i_1, i_2) \rightarrow \mathrm{S}(j_1, j_2) : i_1, i_2 \prec j_1, j_2 \}$ is equal to $\{ \mathrm{S}(i_1, i_2) \rightarrow \mathrm{S}(j_1, j_2) : i_1 < j_1 \vee (i_1 = j_1 \wedge i_2 < j_2) \}$

Syntactic Sugar (2)

- $-e$ is equal to $0 - e$
- $n \cdot e$ is equal to $\underbrace{e + e + \cdots + e}_{n \text{ times}}$ (with n a non-negative integer constant)
- $a_1, \ldots, a_n \prec b_1, \ldots, b_n$ is equal to $\bigvee_{i=1}^{n} \left(\left(\bigwedge_{j=1}^{i-1} a_j = b_j \right) \wedge a_i < b_i \right)$
  Example: $\{ \mathrm{S}(i_1, i_2) \rightarrow \mathrm{S}(j_1, j_2) : i_1, i_2 \prec j_1, j_2 \}$ is equal to
  $\{ \mathrm{S}(i_1, i_2) \rightarrow \mathrm{S}(j_1, j_2) : i_1 < j_1 \vee (i_1 = j_1 \wedge i_2 < j_2) \}$
- $a_1, \ldots, a_n \preccurlyeq b_1, \ldots, b_n$ is equal to
  $a_1, \ldots, a_n \prec b_1, \ldots, b_n \vee a_1, \ldots, a_n = b_1, \ldots, b_n$

Syntactic Sugar (2)

- $-e$ is equal to $0 - e$
- $n \cdot e$ is equal to $\underbrace{e + e + \cdots + e}_{n \text{ times}}$ (with n a non-negative integer constant)
- $a_1, \ldots, a_n \prec b_1, \ldots, b_n$ is equal to $\bigvee_{i=1}^{n} \left(\left(\bigwedge_{j=1}^{i-1} a_j = b_j \right) \wedge a_i < b_i \right)$
  Example: $\{ \mathrm{S}(i_1, i_2) \rightarrow \mathrm{S}(j_1, j_2) : i_1, i_2 \prec j_1, j_2 \}$ is equal to
  $\{ \mathrm{S}(i_1, i_2) \rightarrow \mathrm{S}(j_1, j_2) : i_1 < j_1 \vee (i_1 = j_1 \wedge i_2 < j_2) \}$
- $a_1, \ldots, a_n \preccurlyeq b_1, \ldots, b_n$ is equal to
  $a_1, \ldots, a_n \prec b_1, \ldots, b_n \vee a_1, \ldots, a_n = b_1, \ldots, b_n$
- $a_1, \ldots, a_n \succ b_1, \ldots, b_n$ is equal to $b_1, \ldots, b_n \prec a_1, \ldots, a_n$

Syntactic Sugar (2)

- $-e$ is equal to $0 - e$
- $n \cdot e$ is equal to $\underbrace{e + e + \cdots + e}_{n \text{ times}}$ (with n a non-negative integer constant)
- $a_1, \ldots, a_n \prec b_1, \ldots, b_n$ is equal to $\bigvee_{i=1}^{n} \left(\left(\bigwedge_{j=1}^{i-1} a_j = b_j \right) \wedge a_i < b_i \right)$
  Example: $\{ \mathrm{S}(i_1, i_2) \to \mathrm{S}(j_1, j_2) : i_1, i_2 \prec j_1, j_2 \}$ is equal to
  $\{ \mathrm{S}(i_1, i_2) \to \mathrm{S}(j_1, j_2) : i_1 < j_1 \vee (i_1 = j_1 \wedge i_2 < j_2) \}$
- $a_1, \ldots, a_n \preccurlyeq b_1, \ldots, b_n$ is equal to
  $a_1, \ldots, a_n \prec b_1, \ldots, b_n \vee a_1, \ldots, a_n = b_1, \ldots, b_n$
- $a_1, \ldots, a_n \succ b_1, \ldots, b_n$ is equal to $b_1, \ldots, b_n \prec a_1, \ldots, a_n$
- $a_1, \ldots, a_n \succcurlyeq b_1, \ldots, b_n$ is equal to $b_1, \ldots, b_n \preccurlyeq a_1, \ldots, a_n$

Syntactic Sugar (2)

- $-e$ is equal to $0 - e$
- $n \cdot e$ is equal to $\underbrace{e + e + \cdots + e}_{n \text{ times}}$ (with n a non-negative integer constant)
- $a_1, \ldots, a_n \prec b_1, \ldots, b_n$ is equal to $\bigvee_{i=1}^{n} \left(\left(\bigwedge_{j=1}^{i-1} a_j = b_j \right) \wedge a_i < b_i \right)$
  Example: $\{ \mathrm{S}(i_1, i_2) \to \mathrm{S}(j_1, j_2) : i_1, i_2 \prec j_1, j_2 \}$ is equal to
  $\{ \mathrm{S}(i_1, i_2) \to \mathrm{S}(j_1, j_2) : i_1 < j_1 \vee (i_1 = j_1 \wedge i_2 < j_2) \}$
- $a_1, \ldots, a_n \preccurlyeq b_1, \ldots, b_n$ is equal to
  $a_1, \ldots, a_n \prec b_1, \ldots, b_n \vee a_1, \ldots, a_n = b_1, \ldots, b_n$
- $a_1, \ldots, a_n \succ b_1, \ldots, b_n$ is equal to $b_1, \ldots, b_n \prec a_1, \ldots, a_n$
- $a_1, \ldots, a_n \succcurlyeq b_1, \ldots, b_n$ is equal to $b_1, \ldots, b_n \preccurlyeq a_1, \ldots, a_n$
- a mod n is equal to $a - n \lfloor a/n \rfloor$

Spaces

Recall general form

- Sets

$$\{ S_1(\mathbf{i}) : f_1(\mathbf{i}); S_2(\mathbf{i}) : f_2(\mathbf{i}); \dots \},$$

- Binary relations

$$\{ S_1(\mathbf{i}) \rightarrow T_1(\mathbf{j}) : f_1(\mathbf{i},\mathbf{j}); S_2(\mathbf{i}) \rightarrow T_2(\mathbf{j}) : f_2(\mathbf{i},\mathbf{j}); \dots \}$$

The identifier (e.g., S_1, S_2, T_1, T_2), together with
the dimension, i.e., number of elements in subsequent tuple (e.g., $\mathbf{i}$, $\mathbf{j}$),
will be called a space

When we say $S_2(\mathbf{i}) = T_1(\mathbf{j})$, we mean

- the identifiers S_2 and T_1 are the same
- the dimensions of $\mathbf{i}$ and $\mathbf{j}$ are the same

Examples: $\mathrm{S}() \neq \mathrm{S}(i)$, $\mathrm{S}(a) = \mathrm{S}(b)$, $\mathrm{S}() \neq \mathrm{T}()$

Space Decomposition

General form can be rewritten

$$\{ S_1(\mathbf{i}) \to T_1(\mathbf{j}) : f_1(\mathbf{i},\mathbf{j}); S_2(\mathbf{i}) \to T_2(\mathbf{j}) : f_2(\mathbf{i},\mathbf{j}); \dots \}$$
$$= \bigcup_k \{ S_k(\mathbf{i}) \to T_k(\mathbf{j}) : f_k(\mathbf{i},\mathbf{j}) \}$$

- some operations distribute with union
- other operations are defined on a single (pair of) space(s)
  - $\Rightarrow$ "space local" operations
  - $\Rightarrow$ replace $\{ S(\mathbf{i}) \to T(\mathbf{j}) : f_1(\mathbf{i},\mathbf{j}); S(\mathbf{i}) \to T(\mathbf{j}) : f_2(\mathbf{i},\mathbf{j}) \}$ by $\{ S(\mathbf{i}) \to T(\mathbf{j}) : f_1(\mathbf{i},\mathbf{j}) \vee f_2(\mathbf{i},\mathbf{j}) \}$

In both cases, we define

- unary operator $\oplus$

$$\oplus \bigcup_i R_i := \bigcup_i \oplus R_i$$

- binary operator $\oplus$

$$\left(\bigcup_i R_i \right) \oplus \left(\bigcup_j S_j \right) := \bigcup_i \bigcup_j (R_i \oplus S_j)$$

Basic Operations

$$A = \{\, S(\mathbf{i}_1) \rightarrow T(\mathbf{j}_1) : f(\mathbf{i}_1, \mathbf{j}_1) \,\} \quad B = \{\, U(\mathbf{i}_2) \rightarrow V(\mathbf{j}_2) : g(\mathbf{i}_2, \mathbf{j}_2) \,\}$$

- Union

  $$A \cup B = \{\, S(\mathbf{i}) \rightarrow T(\mathbf{j}) : f(\mathbf{i}, \mathbf{j}); U(\mathbf{i}) \rightarrow V(\mathbf{j}) : g(\mathbf{i}, \mathbf{j}) \,\}$$

- Intersection

  $$A \cap B = \begin{cases} \{\, S(\mathbf{i}) \rightarrow T(\mathbf{j}) : f(\mathbf{i}, \mathbf{j}) \wedge g(\mathbf{i}, \mathbf{j}) \,\} & \text{if } S(\mathbf{i}_1) = U(\mathbf{i}_2) \text{ and } T(\mathbf{j}_1) = V(\mathbf{j}_2) \\ \emptyset & \text{otherwise} \end{cases}$$

- Difference

  $$A \setminus B = \begin{cases} \{\, S(\mathbf{i}) \rightarrow T(\mathbf{j}) : f(\mathbf{i}, \mathbf{j}) \wedge \neg g(\mathbf{i}, \mathbf{j}) \,\} & \text{if } S(\mathbf{i}_1) = U(\mathbf{i}_2) \text{ and } T(\mathbf{j}_1) = V(\mathbf{j}_2) \\ \{\, S(\mathbf{i}) \rightarrow T(\mathbf{j}) : f(\mathbf{i}, \mathbf{j}) \,\} & \text{otherwise} \end{cases}$$

Emptiness Check and Comparisons

- Emptiness check
  Does a set or binary relation contain any elements?
  (for any value of the symbolic constants)

Emptiness Check and Comparisons

- Emptiness check
  Does a set or binary relation contain any elements?
  (for any value of the symbolic constants)
  Is

$$\{(a,b,c,d) : 3d \geq -21 + 19a - 11b - 6c \wedge 3d \leq 21 + 17a - b - 6c \wedge 2b \leq -15 + a \wedge 3d \leq 2 + a + b \wedge 3d \geq a + b\}$$

  empty?

Emptiness Check and Comparisons

- Emptiness check
  Does a set or binary relation contain any elements?
  (for any value of the symbolic constants)
  Is

$$\{(a,b,c,d) : 3d \geq -21 + 19a - 11b - 6c \wedge 3d \leq 21 + 17a - b - 6c \wedge 2b \leq -15 + a \wedge 3d \leq 2 + a + b \wedge 3d \geq a + b\}$$

  empty?

  $\Rightarrow$ No, contains $(13, -1, 38, 4)$ (and infinitely many other elements)

Emptiness Check and Comparisons

- Emptiness check
  Does a set or binary relation contain any elements?
  (for any value of the symbolic constants)
  Is

$$\{(a,b,c,d) : 3d \geq -21 + 19a - 11b - 6c \wedge 3d \leq 21 + 17a - b - 6c \wedge 2b \leq -15 + a \wedge 3d \leq 2 + a + b \wedge 3d \geq a + b\}$$

  empty?

  $\Rightarrow$ No, contains $(13, -1, 38, 4)$ (and infinitely many other elements)

- Comparisons
  - $A \subseteq B$ is defined as $A \setminus B = \emptyset$
  - $A \supseteq B$ is defined as $B \subseteq A$
  - $A = B$ is defined as $A \subseteq B \wedge A \supseteq B$
  - $A \subset B$ is defined as $A \subseteq B \wedge \neg(A = B)$
  - $A \supset B$ is defined as $B \subset A$

Cardinality

- Cardinality of a set
  - $\Rightarrow$ number of elements in the set
  - $\Rightarrow$ may depend on symbolic constants

$S = \{ S(\mathbf{i}) : f(\mathbf{i}) \}$

$\text{card}\, S = \{ n : n = \#\mathbf{i} : f(\mathbf{i}) \}$

$\text{card}\, \{ \mathtt{A}(i) : 0 \leq i \leq n; \mathtt{B}() \} = n + 2$

$$\text{card} \left(\bigcup_i S_i \right) := \sum_i \text{card}\, S_i$$

Cardinality

- Cardinality of a set
  - $\Rightarrow$ number of elements in the set
  - $\Rightarrow$ may depend on symbolic constants

$S = \{ S(\mathbf{i}) : f(\mathbf{i}) \}$

$\operatorname{card} S = \{ n : n = \#\mathbf{i} : f(\mathbf{i}) \}$

$$\operatorname{card}\left(\bigcup_i S_i\right) := \sum_i \operatorname{card} S_i$$

$\operatorname{card} \{ \mathtt{A}(i) : 0 \leq i \leq n; \mathtt{B}() \} = n + 2$

- Cardinality of a binary relation
  - $\Rightarrow$ for each domain element, number of corresponding images

$R = \{ S(\mathbf{i}) \to T(\mathbf{j}) : f(\mathbf{i}, \mathbf{j}) \}$

$\operatorname{card} R = \{ S(\mathbf{i}) \to n : n = \#\mathbf{j} : f(\mathbf{i}, \mathbf{j}) \}$

$$\operatorname{card}\left(\bigcup_i R_i\right) := \sum_i \operatorname{card} R_i$$

$R = \{ \mathtt{A}(i) \to \mathtt{C}(i) : 0 \leq i \leq n; \mathtt{B}() \to \mathtt{C}(i) : 0 \leq i \leq n \}$
$\operatorname{card} R = \{ \mathtt{A}(i) \to 1 : 0 \leq i \leq n; \mathtt{B}() \to n + 1 \}$

Cardinality

- Cardinality of a set
  - $\Rightarrow$ number of elements in the set
  - $\Rightarrow$ may depend on symbolic constants

$S = \{ S(\mathbf{i}) : f(\mathbf{i}) \}$

$\operatorname{card} S = \{ n : n = \#\mathbf{i} : f(\mathbf{i}) \}$

$$\operatorname{card}\left(\bigcup_i S_i\right) := \sum_i \operatorname{card} S_i$$

$\operatorname{card} \{ \mathtt{A}(i) : 0 \leq i \leq n; \mathtt{B}() \} = n + 2$

- Cardinality of a binary relation
  - $\Rightarrow$ for each domain element, number of corresponding images

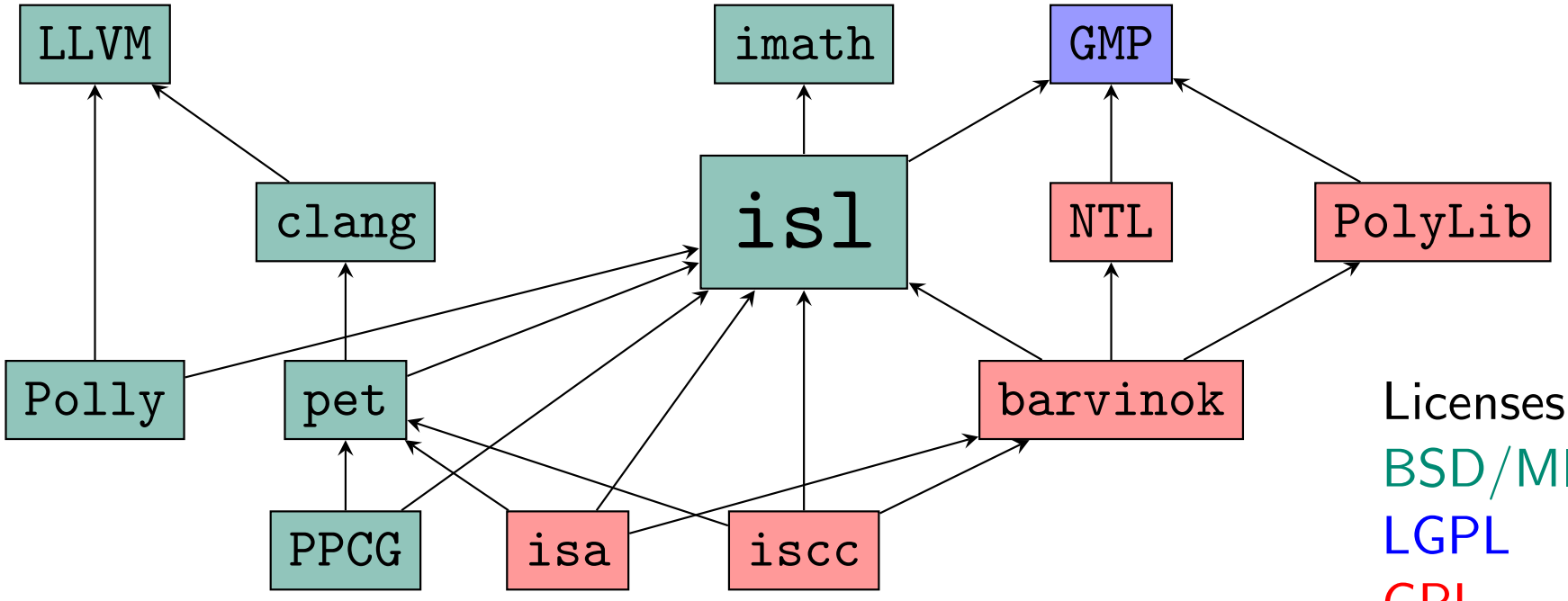
$R = \{ S(\mathbf{i}) \to T(\mathbf{j}) : f(\mathbf{i}, \mathbf{j}) \}$

$\operatorname{card} R = \{ S(\mathbf{i}) \to n : n = \#\mathbf{j} : f(\mathbf{i}, \mathbf{j}) \}$ $\operatorname{card}\left(\bigcup_i R_i\right) := \sum_i \operatorname{card} R_i$

$R = \{ \mathtt{A}(i) \to \mathtt{C}(i) : 0 \leq i \leq n; \mathtt{B}() \to \mathtt{C}(i) : 0 \leq i \leq n \}$
$\operatorname{card} R = \{ \mathtt{A}(i) \to 1 : 0 \leq i \leq n; \mathtt{B}() \to n + 1 \}$

$\Rightarrow$ not a Presburger formula

isl and Related Libraries and Tools



isl: manipulates parametric affine sets and relations

barvinok: counts elements in parametric affine sets and relations

pet: extracts polyhedral model from clang AST

PPCG: Polyhedral Parallel Code Generator

iscc: interactive calculator

isa: prototype tool set including derivation of process networks and equivalence checker

isl/iscc syntax

Relation description

() (tuple)	`[]`
$+, -$	`+, -`
$=, \leq, <, \geq, >$	`=, <=, <, >=, >`
true	`true`
false	`false`
$\wedge$	`and`
$\vee$	`or`
$\neg$	`not`
$\exists v :$	`exists v :`
$\forall v :$	`not exists v : not`
$\preccurlyeq, \prec, \succcurlyeq, \succ$	(not available yet; write out explicitly)

Operations on relations

$\cup$	`+`
$\cap$	`*`
$\setminus$	`-`
$=$	`=`
$\subset, \subseteq, \supset, \supseteq$	`<, <=, >, >=`
card	`card`

Note: symbolic constants need to be explicitly declared

Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

- Number of statement instances

$$\text{card}\{\, \mathtt{S1}(i,j) : 0 \le i < M \wedge 0 \le j < N;$$
$$\mathtt{S2}(i,j,k) : 0 \le i < M \wedge 0 \le j < N \wedge 0 \le k < K \,\}$$

Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

- Number of statement instances

$$\text{card}\{\, \mathtt{S1}(i,j) : 0 \le i < M \wedge 0 \le j < N;$$
$$\mathtt{S2}(i,j,k) : 0 \le i < M \wedge 0 \le j < N \wedge 0 \le k < K \,\}$$

- Number of array elements accessed by each instance

$$\text{card}\{\, \mathtt{S1}(i,j) \to \mathtt{C}(i,j); \mathtt{S2}(i,j,k) \to \mathtt{C}(i,j);$$
$$\mathtt{S2}(i,j,k) \to \mathtt{C}(i,j); \mathtt{S2}(i,j,k) \to \mathtt{A}(i,k); \mathtt{S2}(i,j,k) \to \mathtt{B}(k,j) \,\}$$

Exercise

```
int f1(int m, int n, int A[const static m][n])
{
        int t = 0;
        for (int i = 0; i < m; ++i)
                t += A[i][i];
        return t;
}
void f2(int m, int n, int A[/*.*/m][n][n], int B[/*.*/m][n])
{
        for (int i = 0; i < m; ++i) {
S:              B[i][0] = 0;
                for (int j = 0; j < n; ++j) {
                        if (j == i)
                                continue;
T:                      B[i][j] = f1(j, n, A[i]);
                }
        }
}
```

Exercise

```
int f1(int m, int n, int A[const static m][n])
{
        int t = 0;
        for (int i = 0; i < m; ++i)
                t += A[i][i];
        return t;
}
void f2(int m, int n, int A[/*.*/m][n][n], int B[/*.*/m][n])
{
        for (int i = 0; i < m; ++i) {
S:              B[i][0] = 0;
                for (int j = 0; j < n; ++j) {
                        if (j == i)
                                continue;
T:                      B[i][j] = f1(j, n, A[i]);
                }
        }
}
```

How many statement instances are executed by `f2`?

How many array elements accessed by each instance?

Exercise

```
int f1(int m, int n, int A[const static m][n])
{
        int t = 0;
        for (int i = 0; i < m; ++i)
                t += A[i][i];
        return t;
}
void f2(int m, int n, int A[/*.*/m][n][n], int B[/*.*/m][n])
{
        for (int i = 0; i < m; ++i) {
S:              B[i][0] = 0;
                for (int j = 0; j < n; ++j) {
                        if (j == i)
                                continue;
T:                      B[i][j] = f1(j, n, A[i]);
                }
        }
}
```

How many statement instances are executed by f2? `m>n ? m*n-n+m : m*n`

How many array elements accessed by each instance?

Exercise

```
int f1(int m, int n, int A[const static m][n])
{
        int t = 0;
        for (int i = 0; i < m; ++i)
                t += A[i][i];
        return t;
}
void f2(int m, int n, int A[/*.*/m][n][n], int B[/*.*/m][n])
{
        for (int i = 0; i < m; ++i) {
S:              B[i][0] = 0;
                for (int j = 0; j < n; ++j) {
                        if (j == i)
                                continue;
T:                      B[i][j] = f1(j, n, A[i]);
                }
        }
}
```

How many statement instances are executed by f2? `m>n ? m*n-n+m : m*n`

How many array elements accessed by each instance? `S[i]->1;T[i,j]->1+j`

Outline

Polyhedral Transformation

Two approaches

1. encode execution order in statement instance indices
   $\Rightarrow$ transformation performed by manipulating instance set
2. keep track of execution order separately: schedule
   $\Rightarrow$ transformation performed by manipulating initial/current schedule, or
   $\Rightarrow$ transformation performed by constructing new schedule from scratch

Polyhedral Transformation

Two approaches

1. encode execution order in statement instance indices
   $\Rightarrow$ transformation performed by manipulating instance set
2. keep track of execution order separately: schedule
   $\Rightarrow$ transformation performed by manipulating initial/current schedule, or
   $\Rightarrow$ transformation performed by constructing new schedule from scratch

Schedule O keeps track of relative execution order of statement instances

$\Rightarrow$ for each pair of statement instances $S(\mathbf{i})$ and $T(\mathbf{j})$, schedule determines

- $S(\mathbf{i})$ executed before $T(\mathbf{j})$ $\quad O(S(\mathbf{i}), T(\mathbf{j})) < 0$
- $S(\mathbf{i})$ executed after $T(\mathbf{j})$ $\quad O(S(\mathbf{i}), T(\mathbf{j})) > 0$
- $S(\mathbf{i})$ and $T(\mathbf{j})$ may be executed simultaneously $\quad O(S(\mathbf{i}), T(\mathbf{j})) = 0$

Schedule Representations

Types of schedule representations

- Combined representations
  - schedule tree
  - named Presburger relation
- Scattered representations
  - Kelly's abstraction
  - "$2d + 1$"

[11, 13, 20, 24]

Schedule Representations

Types of schedule representations

- Combined representations
  - schedule tree
  - named Presburger relation
- Scattered representations
  - Kelly's abstraction
  - "$2d + 1$"

Schedule Trees

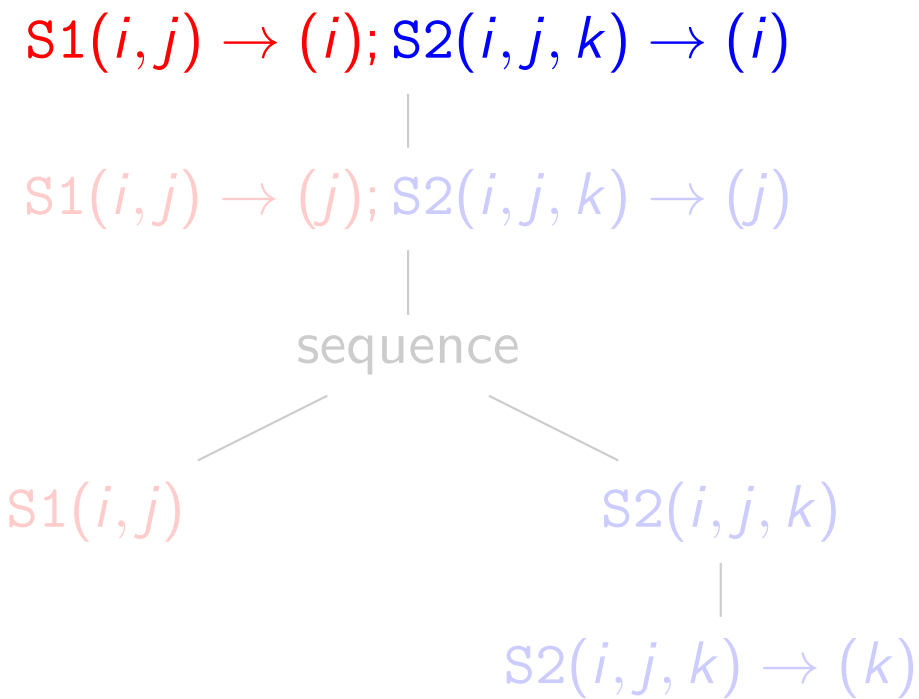
- Main node types
  - sequence: children are executed in order
  - band: instance are executed according to associated piecewise quasi-affine partial schedule P
- Deriving schedule tree from AST
  - `for` loop $\Rightarrow$ single-dimensional band
  - compound statement $\Rightarrow$ sequence

Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```



Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

$$\mathtt{S1}(i,j) \to (i); \mathtt{S2}(i,j,k) \to (i)$$

$$\mathtt{S1}(i,j) \to (j); \mathtt{S2}(i,j,k) \to (j)$$

sequence

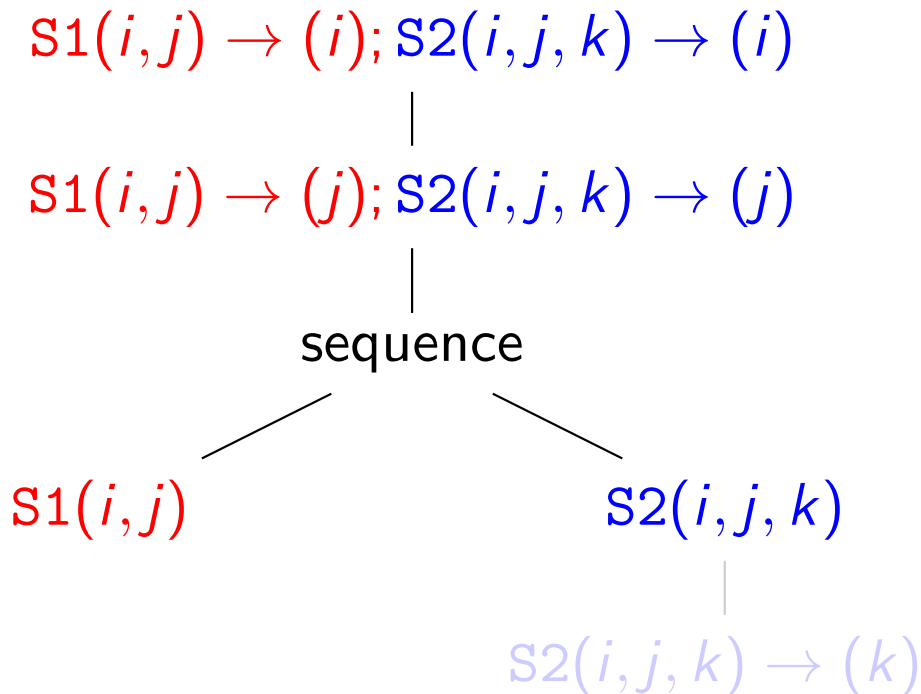
$\mathtt{S1}(i,j)$

$\mathtt{S2}(i,j,k)$

$\mathtt{S2}(i,j,k) \to (k)$

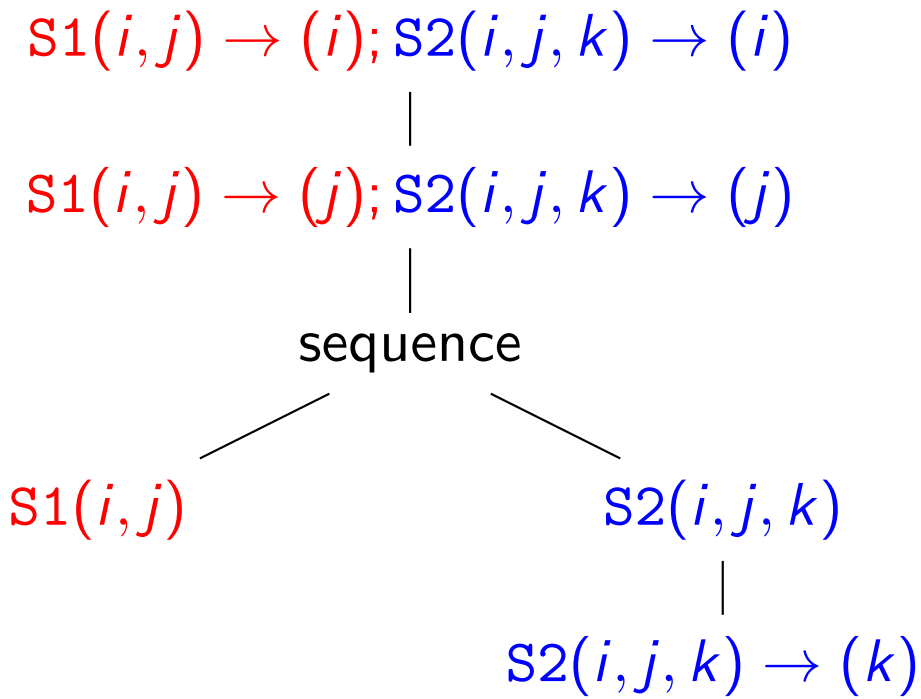
Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```



Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```



Schedule Trees — Execution Order

What is the execution order of statement instances $S(\mathbf{i})$ and $T(\mathbf{j})$ determined by schedule tree O?

Start at root of schedule tree
while *current node is not a leaf* **do**
  if *current node is sequence* **then**
    if $S(\mathbf{i})$ *and* $T(\mathbf{j})$ *appear in same child* **then**
      Move to common child
    else if $S(\mathbf{i})$ *appears in earlier child* **then**
      return $O(S(\mathbf{i}), T(\mathbf{j})) < 0$
    else
      return $O(S(\mathbf{i}), T(\mathbf{j})) > 0$
  else
    if $P(S(\mathbf{i})) = P(T(\mathbf{j}))$ **then**
      Move to single child
    else if $P(S(\mathbf{i})) \prec P(T(\mathbf{j}))$ **then**
      return $O(S(\mathbf{i}), T(\mathbf{j})) < 0$
    else
      return $O(S(\mathbf{i}), T(\mathbf{j})) > 0$
return $O(S(\mathbf{i}), T(\mathbf{j})) = 0$

Named Presburger Relation Schedules

Schedule tree with single (band) node

Named Presburger Relation Schedules

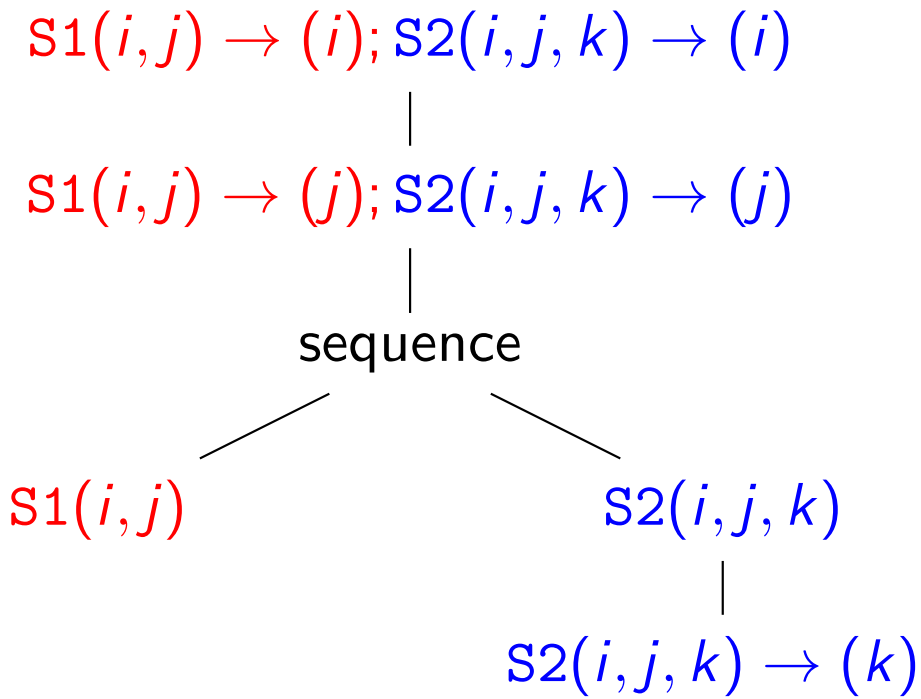
Schedule tree with single (band) node

Flattening a schedule tree

- two nested band nodes
  - $\Rightarrow$ replace by single band node with concatenated partial schedule
- sequence with as children either leaves or trees consisting of a single band node
  - $\Rightarrow$ treat leaves as zero-dimensional band nodes
  - $\Rightarrow$ pad lower-dimensional bands (e.g., with zero)
  - $\Rightarrow$ construct one-dimensional band assigning increasing value to children
  - $\Rightarrow$ combine one-dimensional band with children

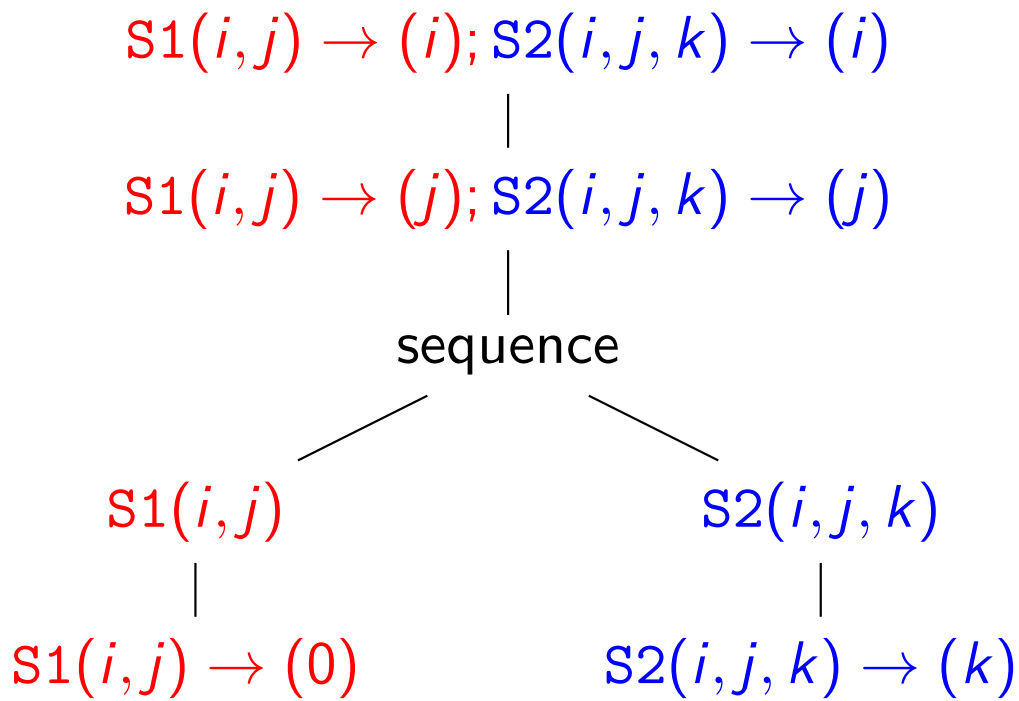
Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```



Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```



Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

$$\begin{array}{c} \mathtt{S1}(i,j) \rightarrow (i); \mathtt{S2}(i,j,k) \rightarrow (i) \\ | \\ \mathtt{S1}(i,j) \rightarrow (j); \mathtt{S2}(i,j,k) \rightarrow (j) \\ | \\ \mathtt{S1}(i,j) \rightarrow (0,0); \mathtt{S2}(i,j,k) \rightarrow (1,k) \end{array}$$

Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

$$\mathtt{S1}(i,j) \to (i); \mathtt{S2}(i,j,k) \to (i)$$

$$\Big\downarrow$$

$$\mathtt{S1}(i,j) \to (j,0,0); \mathtt{S2}(i,j,k) \to (j,1,k)$$

Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

$$\mathtt{S1}(i,j) \rightarrow (i,j,0,0); \mathtt{S2}(i,j,k) \rightarrow (i,j,1,k)$$

Domain and Range of a Relation

$$R = \{ S(\mathbf{i}) \to T(\mathbf{j}) : f(\mathbf{i}, \mathbf{j}) \}$$

- Domain (`iscc: dom`)

  $\mathrm{dom}\, R = \{ S(\mathbf{i}) : \exists \mathbf{j} : f(\mathbf{i}, \mathbf{j}) \}$

  $W = \{ \mathtt{S}(i,j) \to \mathtt{A}(i+j) : 0 \le i < 2 \wedge 0 \le j < 2 \}$
  $\mathrm{dom}\, W = \{ \mathtt{S}(i,j) : 0 \le i < 2 \wedge 0 \le j < 2 \}$

  $\Rightarrow$ statement instances writing any array element

Domain and Range of a Relation

$$R = \{ S(\mathbf{i}) \to T(\mathbf{j}) : f(\mathbf{i}, \mathbf{j}) \}$$

- Domain (`iscc: dom`)

  $\mathrm{dom}\, R = \{ S(\mathbf{i}) : \exists \mathbf{j} : f(\mathbf{i}, \mathbf{j}) \}$

  $W = \{ \mathtt{S}(i,j) \to \mathtt{A}(i+j) : 0 \le i < 2 \wedge 0 \le j < 2 \}$
  $\mathrm{dom}\, W = \{ \mathtt{S}(i,j) : 0 \le i < 2 \wedge 0 \le j < 2 \}$

  $\Rightarrow$ statement instances writing any array element

- Range (`iscc: ran`)

  $\mathrm{ran}\, R = \{ T(\mathbf{j}) : \exists \mathbf{i} : f(\mathbf{i}, \mathbf{j}) \}$

  $W = \{ \mathtt{S}(i,j) \to \mathtt{A}(i+j) : 0 \le i < 2 \wedge 0 \le j < 2 \}$
  $\mathrm{ran}\, W = \{ \mathtt{A}(a) : 0 \le a \le 2 \}$

  $\Rightarrow$ written array elements

Domain/Range Restriction

$$A = \{ S_1(\mathbf{i}_1) : f(\mathbf{i}_1) \} \qquad B = \{ T_1(\mathbf{j}_1) : g(\mathbf{j}_1) \}$$
$$C = \{ S_2(\mathbf{i}_2) \to T_2(\mathbf{j}_2) : h(\mathbf{i}_2, \mathbf{j}_2) \}$$

- Product relation (`iscc: ->`)

  $A \to B = \{ S_1(\mathbf{i}) \to T_1(\mathbf{j}) : f(\mathbf{i}) \wedge g(\mathbf{j}) \}$

Domain/Range Restriction

$$A = \{ S_1(\mathbf{i}_1) : f(\mathbf{i}_1) \} \qquad B = \{ T_1(\mathbf{j}_1) : g(\mathbf{j}_1) \}$$
$$C = \{ S_2(\mathbf{i}_2) \to T_2(\mathbf{j}_2) : h(\mathbf{i}_2, \mathbf{j}_2) \}$$

- Product relation (`iscc: ->`)

  $A \to B = \{ S_1(\mathbf{i}) \to T_1(\mathbf{j}) : f(\mathbf{i}) \wedge g(\mathbf{j}) \}$

- Domain restriction (`iscc: *`)

$$R \cap_{dom} S = R \cap (S \to (\mathrm{ran}\, R))$$

$$C \cap_{dom} A = \begin{cases} \{ S_2(\mathbf{i}) \to T_2(\mathbf{j}) : f(\mathbf{i}) \wedge h(\mathbf{i}, \mathbf{j}) \} & \text{if } S_1(\mathbf{i}_1) = S_2(\mathbf{i}_2) \\ \emptyset & \text{otherwise} \end{cases}$$

Domain/Range Restriction

$$A = \{ S_1(\mathbf{i}_1) : f(\mathbf{i}_1) \} \qquad B = \{ T_1(\mathbf{j}_1) : g(\mathbf{j}_1) \}$$
$$C = \{ S_2(\mathbf{i}_2) \to T_2(\mathbf{j}_2) : h(\mathbf{i}_2, \mathbf{j}_2) \}$$

- Product relation (`iscc: ->`)

  $A \to B = \{ S_1(\mathbf{i}) \to T_1(\mathbf{j}) : f(\mathbf{i}) \wedge g(\mathbf{j}) \}$

- Domain restriction (`iscc: *`)

$$R \cap_{dom} S = R \cap (S \to (\mathrm{ran}\, R))$$

$$C \cap_{dom} A = \begin{cases} \{ S_2(\mathbf{i}) \to T_2(\mathbf{j}) : f(\mathbf{i}) \wedge h(\mathbf{i}, \mathbf{j}) \} & \text{if } S_1(\mathbf{i}_1) = S_2(\mathbf{i}_2) \\ \emptyset & \text{otherwise} \end{cases}$$

- Range restriction

$$R \cap_{ran} S = R \cap ((\mathrm{dom}\, R) \to S)$$

$$C \cap_{ran} A = \begin{cases} \{ S_2(\mathbf{i}) \to T_2(\mathbf{j}) : f(\mathbf{j}) \wedge h(\mathbf{i}, \mathbf{j}) \} & \text{if } S_1(\mathbf{i}_1) = T_2(\mathbf{j}_2) \\ \emptyset & \text{otherwise} \end{cases}$$

AST Generation

[5, 6]

Input:
- instance set
- schedule

Output:
- AST that visits each domain element according to the order specified by the schedule

Note: in case of flat schedule, schedule order is lexicographic order of output space

$\Rightarrow$ single output space

`iscc codegen` operation takes as input flat schedule with instance set encoded in domain

$\Rightarrow$ apply $*$ to "pure" schedule and instance set first

Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

```
I := [M,N,K] -> { S1[i,j] : 0 <= i < M and 0 <= j < N;
  S2[i,j,k] : 0 <= i < M and 0 <= j < N and 0 <= k < K };
O := { S1[i,j] -> [i,j,0,0]; S2[i,j,k] -> [i,j,1,k] };
codegen (O * I);
```

Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

```
I := [M,N,K] -> { S1[i,j] : 0 <= i < M and 0 <= j < N;
  S2[i,j,k] : 0 <= i < M and 0 <= j < N and 0 <= k < K };
O := { S1[i,j] -> [i,j,0,0]; S2[i,j,k] -> [i,j,1,k] };
codegen (O * I);
```

```
for (int c0 = 0; c0 < M; c0 += 1)
  for (int c1 = 0; c1 < N; c1 += 1) {
    S1(c0, c1);
    for (int c3 = 0; c3 < K; c3 += 1)
      S2(c0, c1, c3);
  }
```

Exercise

```
int f1(int m, int n, int A[const static m][n])
{
        int t = 0;
        for (int i = 0; i < m; ++i)
                t += A[i][i];
        return t;
}
void f2(int m, int n, int A[/*.*/m][n][n], int B[/*.*/m][n])
{
        for (int i = 0; i < m; ++i) {
S:              B[i][0] = 0;
                for (int j = 0; j < n; ++j) {
                        if (j == i)
                                continue;
T:                      B[i][j] = f1(j, n, A[i]);
                }
        }
}
```

Write down schedule and generate AST codegen (0 * I)

Outline

Schedule Validity

[1]

Not all schedules correspond to a valid execution order

Schedule Validity

[1]

Not all schedules correspond to a valid execution order

R(a) W(a) $\longrightarrow$ R(a) W(b) W(a) W(a)

Internal restrictions

- A read of a value should not be scheduled after the write of the value
- No other write to same memory location may be scheduled in between

External restrictions (on non-temporaries)

- No write may be scheduled before initial read from a memory location
- No write may be scheduled after last write to a memory location

Schedule Validity

[1]

Not all schedules correspond to a valid execution order

Internal restrictions

- A read of a value should not be scheduled after the write of the value
- No other write to same memory location may be scheduled in between

External restrictions (on non-temporaries)

- No write may be scheduled before initial read from a memory location
- No write may be scheduled after last write to a memory location

Schedule Validity

[1]

Not all schedules correspond to a valid execution order

Internal restrictions

- A read of a value should not be scheduled after the write of the value
- No other write to same memory location may be scheduled in between

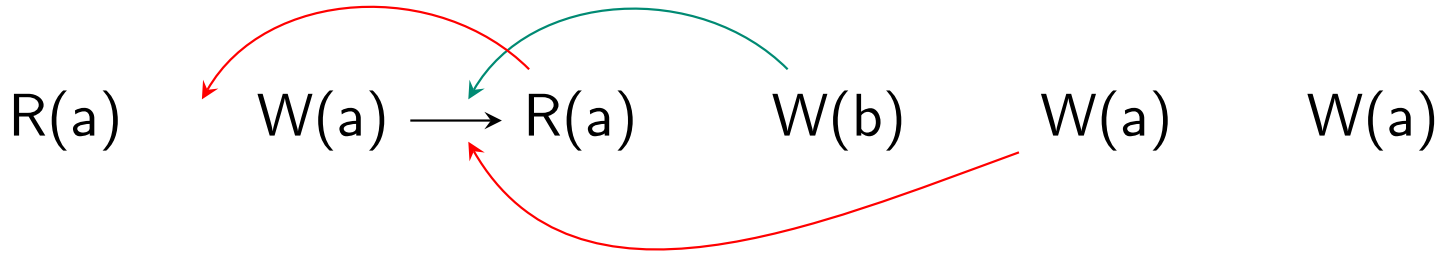
External restrictions (on non-temporaries)

- No write may be scheduled before initial read from a memory location
- No write may be scheduled after last write to a memory location

Schedule Validity

[1]

Not all schedules correspond to a valid execution order



Internal restrictions

- A read of a value should not be scheduled after the write of the value
- No other write to same memory location may be scheduled in between

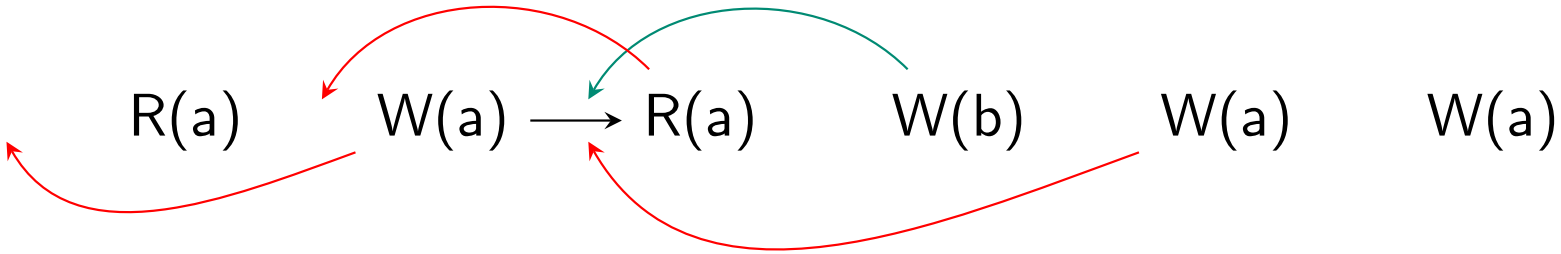
External restrictions (on non-temporaries)

- No write may be scheduled before initial read from a memory location
- No write may be scheduled after last write to a memory location

Schedule Validity

[1]

Not all schedules correspond to a valid execution order



Internal restrictions

- A read of a value should not be scheduled after the write of the value
- No other write to same memory location may be scheduled in between

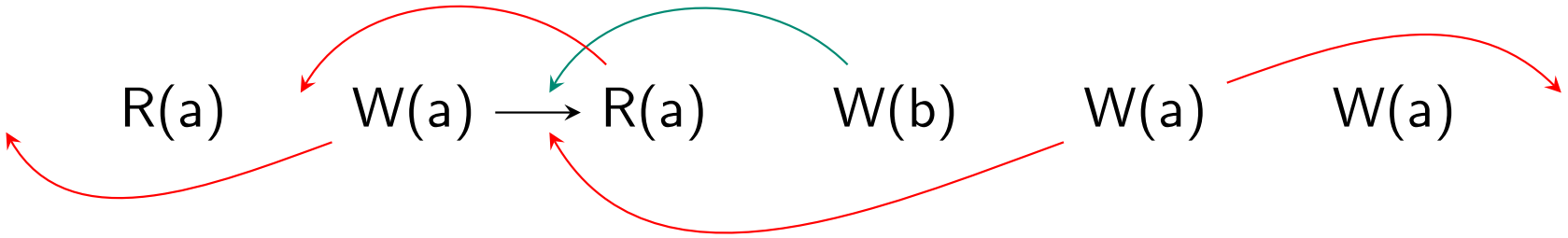
External restrictions (on non-temporaries)

- No write may be scheduled before initial read from a memory location
- No write may be scheduled after last write to a memory location

Schedule Validity

[1]

Not all schedules correspond to a valid execution order



Internal restrictions

- A read of a value should not be scheduled after the write of the value
- No other write to same memory location may be scheduled in between

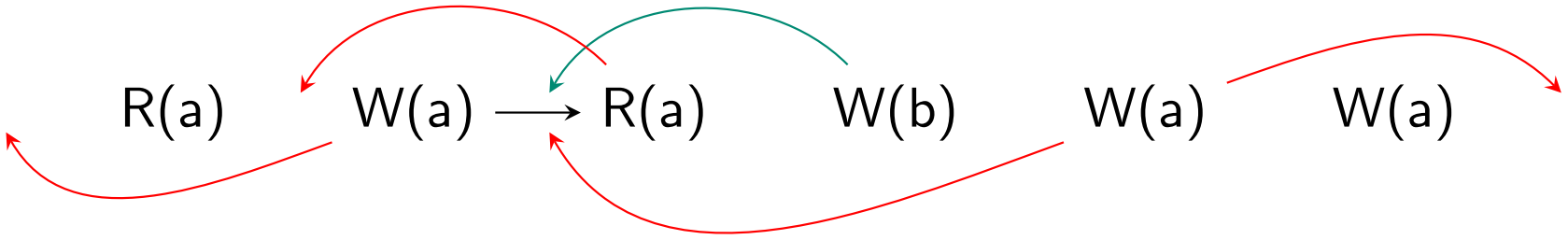
External restrictions (on non-temporaries)

- No write may be scheduled before initial read from a memory location
- No write may be scheduled after last write to a memory location

Schedule Validity

[1]

Not all schedules correspond to a valid execution order



Internal restrictions

- A read of a value should not be scheduled after the write of the value
- No other write to same memory location may be scheduled in between

External restrictions (on non-temporaries)

- No write may be scheduled before initial read from a memory location
- No write may be scheduled after last write to a memory location

Sufficient conditions:

- Every read of a memory location is scheduled after every previous write to the same memory location
- Every write to a memory location is scheduled after every previous read or write to the same memory location

Dependences

Sufficient conditions for schedule validity:

- Every read of a memory location is scheduled after every previous write to the same memory location
- Every write to a memory location is scheduled after every previous read or write to the same memory location

Dependence relation D: pairs of statement instances

- accessing the same memory location
- of which at least one is a write
- with the first executed before the second

Sufficient condition:

$$\forall S(\mathbf{i}) \rightarrow T(\mathbf{j}) \in D : O(S(\mathbf{i}), T(\mathbf{j})) < 0$$

Inverse Relation and Composition

$$A = \{ S_1(\mathbf{i}_1) \to T_1(\mathbf{j}_1) : f(\mathbf{i}_1, \mathbf{j}_1) \} \qquad B = \{ S_2(\mathbf{i}_2) \to T_2(\mathbf{j}_2) : g(\mathbf{i}_2, \mathbf{j}_2) \}$$

- Inverse (`iscc: ^-1`)

  $A^{-1} = \{ T_1(\mathbf{j}) \to S_1(\mathbf{i}) : f(\mathbf{i}, \mathbf{j}) \}$

  $W = \{ \mathtt{S}(i,j) \to \mathtt{A}(i+j) : 0 \le i < 2 \wedge 0 \le j < 2 \}$
  $W^{-1} = \{ \mathtt{A}(a) \to \mathtt{S}(i,j) : a = i + j \wedge 0 \le i < 2 \wedge 0 \le j < 2 \}$

  $\Rightarrow$ statement instances writing array element

Inverse Relation and Composition

$$A = \{ S_1(\mathbf{i}_1) \to T_1(\mathbf{j}_1) : f(\mathbf{i}_1, \mathbf{j}_1) \} \qquad B = \{ S_2(\mathbf{i}_2) \to T_2(\mathbf{j}_2) : g(\mathbf{i}_2, \mathbf{j}_2) \}$$

- Inverse (`iscc: ^-1`)

  $A^{-1} = \{ T_1(\mathbf{j}) \to S_1(\mathbf{i}) : f(\mathbf{i}, \mathbf{j}) \}$

  $W = \{ \mathtt{S}(i,j) \to \mathtt{A}(i+j) : 0 \le i < 2 \wedge 0 \le j < 2 \}$
  $W^{-1} = \{ \mathtt{A}(a) \to \mathtt{S}(i,j) : a = i + j \wedge 0 \le i < 2 \wedge 0 \le j < 2 \}$

  $\Rightarrow$ statement instances writing array element

- Composition (`iscc: after`)

$$B \circ A = \begin{cases} \{ S_1(\mathbf{i}) \to T_2(\mathbf{j}) : \exists \mathbf{k} : f(\mathbf{i}, \mathbf{k}) \wedge g(\mathbf{k}, \mathbf{j}) \} & \text{if } T_1(\mathbf{j}_1) = S_2(\mathbf{i}_2) \\ \emptyset & \text{otherwise} \end{cases}$$

  $W^{-1} \circ W = \{ \mathtt{S}(i,j) \to \mathtt{S}(i',j') : 0 \le i, i', j, j' < 2 \wedge i + j = i' + j' \}$

  $\Rightarrow$ pairs of statement instances that write same array element

Lexicographic Order

- Sets (`iscc: <<`)

$$A = \{\, S(\mathbf{i}) : f(\mathbf{i}) \,\}$$

$$B = \{\, T(\mathbf{j}) : g(\mathbf{j}) \,\}$$

$$A \prec B = \begin{cases} \{\, S(\mathbf{i}) \rightarrow S(\mathbf{j}) : f(\mathbf{i}) \wedge g(\mathbf{j}) \wedge \mathbf{i} \prec \mathbf{j} \,\} & \text{if } S(\mathbf{i}) = T(\mathbf{j}) \\ \emptyset & \text{otherwise} \end{cases}$$

Lexicographic Order

- Sets (`iscc: <<`)

$$A = \{ S(\mathbf{i}) : f(\mathbf{i}) \}$$
$$B = \{ T(\mathbf{j}) : g(\mathbf{j}) \}$$
$$A \prec B = \begin{cases} \{ S(\mathbf{i}) \to S(\mathbf{j}) : f(\mathbf{i}) \wedge g(\mathbf{j}) \wedge \mathbf{i} \prec \mathbf{j} \} & \text{if } S(\mathbf{i}) = T(\mathbf{j}) \\ \emptyset & \text{otherwise} \end{cases}$$

- Relations (`iscc: <<`)

$\Rightarrow$ binary relation on domains reflecting lexicographic order of images

$$A = \{ S_1(\mathbf{i}_1) \to T_1(\mathbf{j}_1) : f(\mathbf{i}_1, \mathbf{j}_1) \}$$
$$B = \{ S_2(\mathbf{i}_2) \to T_2(\mathbf{j}_2) : g(\mathbf{i}_2, \mathbf{j}_2) \}$$
$$A \prec B = \begin{cases} \{ S_1(\mathbf{i}_1) \to S_2(\mathbf{i}_2) : \exists \mathbf{j}_1, \mathbf{j}_2 : f(\mathbf{i}_1, \mathbf{j}_1) \wedge g(\mathbf{i}_2, \mathbf{j}_2) \wedge \mathbf{j}_1 \prec \mathbf{j}_2 \} & \text{if } T_1(\mathbf{j}_1) = T_2(\mathbf{j}_2) \\ \emptyset & \text{otherwise} \end{cases}$$

Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

```
O := { S1[i,j] -> [i,j,0,0]; S2[i,j,k] -> [i,j,1,k] };
O << O;
```

Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

```
O := { S1[i,j] -> [i,j,0,0]; S2[i,j,k] -> [i,j,1,k] };
O << O;
```

```
{ S2[i, j, k] -> S2[i', j', k'] : i' >= 1 + i;
  S2[i, j, k] -> S2[i, j', k'] : j' >= 1 + j;
  S2[i, j, k] -> S2[i, j, k'] : k' >= 1 + k;
  S1[i, j] -> S2[i', j', k] : i' >= 1 + i;
  S1[i, j] -> S2[i, j', k] : j' >= 1 + j;
  S1[i, j] -> S2[i, j, k]; S2[i, j, k] -> S1[i', j'] : i' >= 1 + i;
  S2[i, j, k] -> S1[i, j'] : j' >= 1 + j;
  S1[i, j] -> S1[i', j'] : i' >= 1 + i;
  S1[i, j] -> S1[i, j'] : j' >= 1 + j }
```

Dependence Analysis

Recall: sufficient condition for schedule validity

$$\forall S(\mathbf{i}) \rightarrow T(\mathbf{j}) \in D : O(S(\mathbf{i}), T(\mathbf{j})) < 0$$

Dependence relation D: pairs of statement instances

- accessing the same memory location
- of which at least one is a write
- with the first executed before the second

Dependence Analysis

Recall: sufficient condition for schedule validity

$$\forall S(\mathbf{i}) \rightarrow T(\mathbf{j}) \in D : O(S(\mathbf{i}), T(\mathbf{j})) < 0$$

Dependence relation D: pairs of statement instances

- accessing the same memory location
- of which at least one is a write
- with the first executed before the second

Computation:

$$D = \left(\left(W^{-1} \circ R\right) \cup \left(W^{-1} \circ W\right) \cup \left(R^{-1} \circ W\right)\right) \cap \left(O \prec O\right)$$

W: write access relation

R: read access relation

O: original schedule

Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

```
W := { S1[i,j] -> C[i,j]; S2[i,j,k] -> C[i,j] };
R := { S2[i,j,k] -> C[i,j]; S2[i,j,k] -> A[i,k];
       S2[i,j,k] -> B[k,j] };
I := [M,N,K] -> { S1[i,j] : 0 <= i < M and 0 <= j < N;
      S2[i,j,k] : 0 <= i < M and 0 <= j < N and 0 <= k < K };
O := { S1[i,j] -> [i,j,0,0]; S2[i,j,k] -> [i,j,1,k] };
((R . W^-1) + (W . W^-1) + (W . R^-1)) * (O << O);
```

Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

```
W := { S1[i,j] -> C[i,j]; S2[i,j,k] -> C[i,j] };
R := { S2[i,j,k] -> C[i,j]; S2[i,j,k] -> A[i,k];
       S2[i,j,k] -> B[k,j] };
I := [M,N,K] -> { S1[i,j] : 0 <= i < M and 0 <= j < N;
      S2[i,j,k] : 0 <= i < M and 0 <= j < N and 0 <= k < K };
O := { S1[i,j] -> [i,j,0,0]; S2[i,j,k] -> [i,j,1,k] };
((R . W^-1) + (W . W^-1) + (W . R^-1)) * (O << O);
```

```
{ S2[i, j, k] -> S2[i, j, k'] : k' >= 1 + k;
  S1[i, j] -> S2[i, j, k] }
```

Exercise

```
int f1(int m, int n, int A[const static m][n])
{
        int t = 0;
        for (int i = 0; i < m; ++i)
                t += A[i][i];
        return t;
}
void f2(int m, int n, int A[/*.*/m][n][n], int B[/*.*/m][n])
{
        for (int i = 0; i < m; ++i) {
S:              B[i][0] = 0;
                for (int j = 0; j < n; ++j) {
                        if (j == i)
                                continue;
T:                      B[i][j] = f1(j, n, A[i]);
                }
        }
}
```

Compute dependence relation

Exercise

```
int f1(int m, int n, int A[const static m][n])
{
        int t = 0;
        for (int i = 0; i < m; ++i)
                t += A[i][i];
        return t;
}
void f2(int m, int n, int A[/*.*/m][n][n], int B[/*.*/m][n])
{
        for (int i = 0; i < m; ++i) {
S:              B[i][0] = 0;
                for (int j = 0; j < n; ++j) {
                        if (j == i)
                                continue;
T:                      B[i][j] = f1(j, n, A[i]);
                }
        }
}
```

Compute dependence relation {}

Accesses to Structure Fields and Nested Relations

No special treatment is needed for representing accesses to structure fields

$\Rightarrow$ structure field encoded in name of target space of access relations

Accesses to Structure Fields and Nested Relations

No special treatment is needed for representing accesses to structure fields

$\Rightarrow$ structure field encoded in name of target space of access relations

```
struct s {
        int a;
        int b[10];
};

void f(struct s s[const static 10][10])
{
        for (int i = 0; i < 10; ++i)
S:              s[i][i].b[9 - i] = 0;
}
```

$\{ \mathrm{S}(i) \rightarrow \mathrm{s\_b}(i, i, 9 - i) \}$

Accesses to Structure Fields and Nested Relations

No special treatment is needed for representing accesses to structure fields

$\Rightarrow$ structure field encoded in name and/or structure of target space of access relations

```
struct s {
        int a;
        int b[10];
};

void f(struct s s[const static 10][10])
{
        for (int i = 0; i < 10; ++i)
S:              s[i][i].b[9 - i] = 0;
}
```

$\{ \mathrm{S}(i) \rightarrow \mathrm{s\_b}(i, i, 9 - i) \}$

In pet, structure encoded in *nested relation*

$\{ \mathrm{S}(i) \rightarrow \mathrm{s\_b}(\mathrm{s}(i, i) \rightarrow \mathrm{b}(9 - i)) \}$

Accesses to Structures

Dependence analysis needs to take into account that access to structure represents access to all fields of structure

```
struct c {
        float re;
        float im;
};

void f(struct c A[const static 10])
{
S:      A[0] = A[2];
T:      A[1].re = A[0].im;
}
```

Write access relation: $\{ \mathtt{S}() \rightarrow \mathtt{A}(0); \mathtt{T}() \rightarrow \mathtt{A\_re}(\mathtt{A}(1) \rightarrow \mathtt{re}()) \}$

Accesses to Structures

Dependence analysis needs to take into account that access to structure represents access to all fields of structure

```
struct c {
        float re;
        float im;
};

void f(struct c A[const static 10])
{
S:      A[0] = A[2];
T:      A[1].re = A[0].im;
}
```

Write access relation: $\{ \mathtt{S}() \to \mathtt{A}(0); \mathtt{T}() \to \mathtt{A\_re}(\mathtt{A}(1) \to \mathtt{re}()) \}$

Expansion: $\{ \mathtt{A}(a) \to \mathtt{A\_re}(\mathtt{A}(a) \to \mathtt{re}()); \mathtt{A}(a) \to \mathtt{A\_im}(\mathtt{A}(a) \to \mathtt{im}()); \}$

Expanded write access relation:
$\{ \mathtt{S}() \to \mathtt{A\_re}(\mathtt{A}(0) \to \mathtt{re}()); \mathtt{S}() \to \mathtt{A\_im}(\mathtt{A}(0) \to \mathtt{im}());$
$\mathtt{T}() \to \mathtt{A\_re}(\mathtt{A}(1) \to \mathtt{re}()) \}$

Outline

Schedule Optimization Criteria

Typical optimization criteria

- increase parallelism
- increase locality
- reduce memory requirements

Schedule Optimization Criteria

Typical optimization criteria

- increase parallelism
- increase locality
- reduce memory requirements

Parallelism:
Pairs of statement instances $S(\mathbf{i})$ and $T(\mathbf{j})$ may be executed in parallel if they do not depend on each other:

$$\{ S(\mathbf{i}) \rightarrow T(\mathbf{j}); T(\mathbf{j}) \rightarrow S(\mathbf{i}) \} \cap D = \emptyset$$

Local Parallelism

Global parallelism

$$\{ S(\mathbf{i}) \to T(\mathbf{j}); T(\mathbf{j}) \to S(\mathbf{i}) \} \cap D = \emptyset$$

Local parallelism

$$\{ S(\mathbf{i}) \to T(\mathbf{j}); T(\mathbf{j}) \to S(\mathbf{i}) \} \cap L = \emptyset$$

- Root of schedule tree: $L = D$
- Child of band node b with partial schedule P:
  $L = L_b \cap \{ S(\mathbf{i}) \to T(\mathbf{j}) : P(S(\mathbf{i})) = P(T(\mathbf{j})) \}$
- Child of sequence node s with instance set F
  $L = L_s \cap \{ S(\mathbf{i}) \to T(\mathbf{j}) : S(\mathbf{i}) \in F \wedge T(\mathbf{j}) \in F \}$

Encoding Parallelism

- Coarse grain parallelism (root)
  - *placement* maps statement instances to virtual processors
  - schedule interpreted within each virtual processor

Encoding Parallelism

- Coarse grain parallelism (root)
  - *placement* maps statement instances to virtual processors
  - schedule interpreted within each virtual processor
- Fine grain parallelism (leaf)
  - statement instances $S(\mathbf{i})$ and $T(\mathbf{j})$ for which $O(S(\mathbf{i}), T(\mathbf{j})) = 0$ may be executed in parallel within leaf that contains both

Encoding Parallelism

- Coarse grain parallelism (root)
  - *placement* maps statement instances to virtual processors
  - schedule interpreted within each virtual processor
- Fine grain parallelism (leaf)
  - statement instances $S(\mathbf{i})$ and $T(\mathbf{j})$ for which $O(S(\mathbf{i}), T(\mathbf{j})) = 0$ may be executed in parallel within leaf that contains both
- General case (arbitrary position in schedule tree)
  - schedules defines total order, but,
  - some sequence nodes and/or some band dimensions are explicitly marked parallel

False Dependences

```
for (int i = 0; i < n; ++i) {
S:      t = f1(A[i]);
T:      B[i] = f2(t);
}
```

Dependences

- read after write ("true"): $\{ \mathtt{S}(i) \rightarrow \mathtt{T}(i') : i' \geq i \}$
- write after read ("anti"): $\{ \mathtt{T}(i) \rightarrow \mathtt{S}(i') : i' > i \}$
- write after write ("output"): $\{ \mathtt{S}(i) \rightarrow \mathtt{S}(i') : i' > i \}$

False Dependences

```
for (int i = 0; i < n; ++i) {
S:      t = f1(A[i]);
T:      B[i] = f2(t);
}
```

Dependences

- read after write ("true"): $\{ \mathtt{S}(i) \to \mathtt{T}(i') : i' \geq i \}$
- write after read ("anti"): $\{ \mathtt{T}(i) \to \mathtt{S}(i') : i' > i \}$
- write after write ("output"): $\{ \mathtt{S}(i) \to \mathtt{S}(i') : i' > i \}$

} "false"

False Dependences

```
for (int i = 0; i < n; ++i) {
S:      t = f1(A[i]);
T:      B[i] = f2(t);
}
```

Dependences

- read after write ("true"): $\{ \mathtt{S}(i) \to \mathtt{T}(i') : i' \geq i \}$
- write after read ("anti"): $\{ \mathtt{T}(i) \to \mathtt{S}(i') : i' > i \}$
- write after write ("output"): $\{ \mathtt{S}(i) \to \mathtt{S}(i') : i' > i \}$

} "false"

False dependences not from dataflow, but from reuse of memory location `t`

Possible solution: expansion/privatization

```
for (int i = 0; i < n; ++i) {
S:      t[i] = f1(A[i]);
T:      B[i] = f2(t[i]);
}
```

- dataflow (subset of "true" dependences): $\{ \mathtt{S}(i) \to \mathtt{T}(i) \}$

False Dependences

```
for (int i = 0; i < n; ++i) {
S:      t = f1(A[i]);
T:      B[i] = f2(t);
}
```

Dependences

- read after write ("true"): $\{ \mathtt{S}(i) \to \mathtt{T}(i') : i' \geq i \}$
- write after read ("anti"): $\{ \mathtt{T}(i) \to \mathtt{S}(i') : i' > i \}$
- write after write ("output"): $\{ \mathtt{S}(i) \to \mathtt{S}(i') : i' > i \}$

} "false"

False dependences not from dataflow, but from reuse of memory location `t`

Possible solution: expansion/privatization

```
for (int i = 0; i < n; ++i) {
S:      t[i] = f1(A[i]);
T:      B[i] = f2(t[i]);
}
```

- dataflow (subset of "true" dependences): $\{ \mathtt{S}(i) \to \mathtt{T}(i) \}$

Lexicographic Optimization (Space Local)

- Lexicographic Minimum of Sets (`iscc: lexmin`)

  $S = \{ S(\mathbf{i}) : f(\mathbf{i}) \}$

  $\text{lexmin}\, S = \{ S(\mathbf{i}) : f(\mathbf{i}) \wedge \forall \mathbf{i}' : f(\mathbf{i}') \Rightarrow \mathbf{i} \preccurlyeq \mathbf{i}' \}$

  $I = \{ \mathtt{R}(); \mathtt{S}(i,j) : 0 \leq i < 2 \wedge 0 \leq j < 2; \mathtt{T}(k) : 0 \leq k < 2 \}$

  $\text{lexmin}\, I = \{ \mathtt{R}(); \mathtt{S}(0,0); \mathtt{T}(0) \}$

Lexicographic Optimization (Space Local)

- Lexicographic Minimum of Sets (iscc: `lexmin`)

  $S = \{ S(\mathbf{i}) : f(\mathbf{i}) \}$
  $\operatorname{lexmin} S = \{ S(\mathbf{i}) : f(\mathbf{i}) \wedge \forall \mathbf{i}' : f(\mathbf{i}') \Rightarrow \mathbf{i} \preccurlyeq \mathbf{i}' \}$

  $I = \{ \mathtt{R}(); \mathtt{S}(i,j) : 0 \leq i < 2 \wedge 0 \leq j < 2; \mathtt{T}(k) : 0 \leq k < 2 \}$
  $\operatorname{lexmin} I = \{ \mathtt{R}(); \mathtt{S}(0,0); \mathtt{T}(0) \}$

- Lexicographic Maximum of Sets (iscc: `lexmax`)

  $S = \{ S(\mathbf{i}) : f(\mathbf{i}) \}$
  $\operatorname{lexmax} S = \{ S(\mathbf{i}) : f(\mathbf{i}) \wedge \forall \mathbf{i}' : f(\mathbf{i}') \Rightarrow \mathbf{i} \succcurlyeq \mathbf{i}' \}$

  $\operatorname{lexmax} I = \{ \mathtt{R}(); \mathtt{S}(1,1); \mathtt{T}(1) \}$

Lexicographic Optimization (Space Local)

- Lexicographic Minimum of Sets (iscc: `lexmin`)

  $S = \{ S(\mathbf{i}) : f(\mathbf{i}) \}$
  $\operatorname{lexmin} S = \{ S(\mathbf{i}) : f(\mathbf{i}) \wedge \forall \mathbf{i}' : f(\mathbf{i}') \Rightarrow \mathbf{i} \preccurlyeq \mathbf{i}' \}$

  $I = \{ \mathtt{R}(); \mathtt{S}(i,j) : 0 \le i < 2 \wedge 0 \le j < 2; \mathtt{T}(k) : 0 \le k < 2 \}$
  $\operatorname{lexmin} I = \{ \mathtt{R}(); \mathtt{S}(0,0); \mathtt{T}(0) \}$

- Lexicographic Maximum of Sets (iscc: `lexmax`)

  $S = \{ S(\mathbf{i}) : f(\mathbf{i}) \}$
  $\operatorname{lexmax} S = \{ S(\mathbf{i}) : f(\mathbf{i}) \wedge \forall \mathbf{i}' : f(\mathbf{i}') \Rightarrow \mathbf{i} \succcurlyeq \mathbf{i}' \}$

  $\operatorname{lexmax} I = \{ \mathtt{R}(); \mathtt{S}(1,1); \mathtt{T}(1) \}$

- Lexicographic Maximum of Relations (iscc: `lexmax`)

  $R = \{ S(\mathbf{i}) \to T(\mathbf{j}) : f(\mathbf{i},\mathbf{j}) \}$
  $\operatorname{lexmax} R = \{ S(\mathbf{i}) \to T(\mathbf{j}) : f(\mathbf{i},\mathbf{j}) \wedge \forall \mathbf{j}' : f(\mathbf{i},\mathbf{j}') \Rightarrow \mathbf{j} \succcurlyeq \mathbf{j}' \}$

  $W^{-1} = \{ \mathtt{A}(a) \to \mathtt{S}(i,j) : a = i + j \wedge 0 \le i < 2 \wedge 0 \le j < 2 \}$
  $\operatorname{lexmax}(W^{-1}) = \{ \mathtt{A}(a) \to \mathtt{S}(a,0) : 0 \le a \le 1; \mathtt{A}(2) \to \mathtt{S}(1,1) \}$

  $\Rightarrow$ last statement instance writing array element

Array Dataflow Analysis

Given a read from an array element, what was the last write to the same array element before the read?

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
F:      a[i+j] = f(a[i+j]);
for (i = 0; i < N; ++i)
W:  Write(a[i]);
```

Array Dataflow Analysis

Given a read from an array element, what was the last write to the same array element before the read?

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
F:      a[i+j] = f(a[i+j]);
for (i = 0; i < N; ++i)
W:  Write(a[i]);
```

Array Dataflow Analysis

Given a read from an array element, what was the last write to the same array element before the read?

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
F:      a[i+j] = f(a[i+j]);
for (i = 0; i < N; ++i)
W:  Write(a[i]);
```

Access relations:

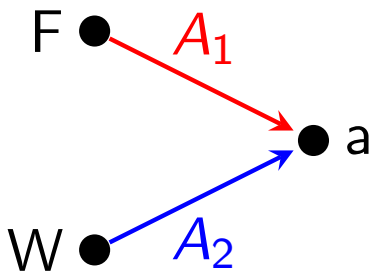
$A_1 = \{ \mathtt{F}(i,j) \rightarrow \mathtt{a}(i+j) : 0 \leq i < N \wedge 0 \leq j < N - i \}$

$A_2 = \{ \mathtt{W}(i) \rightarrow \mathtt{a}(i) : 0 \leq i < N \}$

Array Dataflow Analysis

Given a read from an array element, what was the last write to the same array element before the read?

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
F:      a[i+j] = f(a[i+j]);
for (i = 0; i < N; ++i)
W:  Write(a[i]);
```



Access relations:

$A_1 = \{ \mathtt{F}(i,j) \rightarrow \mathtt{a}(i+j) : 0 \leq i < N \wedge 0 \leq j < N - i \}$

$A_2 = \{ \mathtt{W}(i) \rightarrow \mathtt{a}(i) : 0 \leq i < N \}$

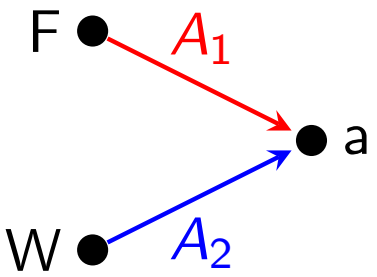
Map to all writes:

$R' = {A_1}^{-1} \circ A_2 = \{ \mathtt{W}(i) \rightarrow \mathtt{F}(i', i - i') : 0 \leq i' \leq i < N \}$

Array Dataflow Analysis

Given a read from an array element, what was the last write to the same array element before the read?

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
F:      a[i+j] = f(a[i+j]);
for (i = 0; i < N; ++i)
W:  Write(a[i]);
```



Access relations:

$A_1 = \{ \mathtt{F}(i,j) \to \mathtt{a}(i+j) : 0 \le i < N \wedge 0 \le j < N - i \}$

$A_2 = \{ \mathtt{W}(i) \to \mathtt{a}(i) : 0 \le i < N \}$

Map to all writes:

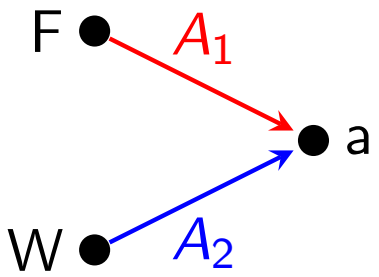
$R' = {A_1}^{-1} \circ A_2 = \{ \mathtt{W}(i) \to \mathtt{F}(i', i - i') : 0 \le i' \le i < N \}$

Last write: $R = \operatorname{lexmax} R' = \{ \mathtt{W}(i) \to \mathtt{F}(i, 0) : 0 \le i < N \}$

Array Dataflow Analysis

Given a read from an array element, what was the last write to the same array element before the read?

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
F:      a[i+j] = f(a[i+j]);
for (i = 0; i < N; ++i)
W:  Write(a[i]);
```



Access relations:

$A_1 = \{ \mathtt{F}(i,j) \to \mathtt{a}(i+j) : 0 \le i < N \wedge 0 \le j < N - i \}$

$A_2 = \{ \mathtt{W}(i) \to \mathtt{a}(i) : 0 \le i < N \}$

Map to all writes:

$R' = {A_1}^{-1} \circ A_2 = \{ \mathtt{W}(i) \to \mathtt{F}(i', i - i') : 0 \le i' \le i < N \}$

Last write: $R = \operatorname{lexmax} R' = \{ \mathtt{W}(i) \to \mathtt{F}(i, 0) : 0 \le i < N \}$

In general: impose lexicographical order on shared branch of schedule tree

Expansion

Assume:

- instance sets and access relations are static and exact
  $\Rightarrow$ each read has exactly one corresponding write
- single read and write per statement
  $\Rightarrow$ expanded array indexed by statement instance of write

Expansion

Assume:

- instance sets and access relations are static and exact
  $\Rightarrow$ each read has exactly one corresponding write
- single read and write per statement
  $\Rightarrow$ expanded array indexed by statement instance of write

```
for (int i = 0; i < n; ++i) {
S:        t = f1(A[i]);
T:        B[i] = f2(t);
}
```

Dataflow: $\{ \mathtt{S}(i) \rightarrow \mathtt{T}(i) \}$

Expansion

Assume:

- instance sets and access relations are static and exact
  $\Rightarrow$ each read has exactly one corresponding write
- single read and write per statement
  $\Rightarrow$ expanded array indexed by statement instance of write

```
for (int i = 0; i < n; ++i) {
S:      t = f1(A[i]);
T:      B[i] = f2(t);
}
```

Dataflow: $\{ \mathtt{S}(i) \rightarrow \mathtt{T}(i) \}$

```
for (int i = 0; i < n; ++i) {
S:      S[i] = f1(A[i]);
T:      B[i] = f2(S[i]);
}
```

$\Rightarrow$ only remaining dependences are dataflow induced

Tagged Access Relations

Expansion in case of multiple reads or writes per statement

$\Rightarrow$ statement instance not enough to identify memory access

$\Rightarrow$ use identifier of array reference instead

Tagged Access Relations

Expansion in case of multiple reads or writes per statement

$\Rightarrow$ statement instance not enough to identify memory access

$\Rightarrow$ use identifier of array reference instead

$\Rightarrow$ `pet`: embed array reference in "tagged" instance set

$\Rightarrow$ "ternary" access relations

$$I \to R \to A$$

I: statement instance
R: reference identifier
O: array element

$\Rightarrow$ in practice: nested binary relation

$$(I \to R) \to A$$

For example, $\{ (\mathtt{S}(i) \to \mathtt{R1}()) \to \mathtt{A}(i) \}$

Wrapping, Unwrapping, Domain Map and Range Map

$$R = \{ S(\mathbf{i}) \to T(\mathbf{j}) : f(\mathbf{i},\mathbf{j}) \} \qquad S = \{ (U(\mathbf{i}) \to V(\mathbf{j})) : g(\mathbf{i},\mathbf{j}) \}$$

- Wrap (`iscc: wrap`)

  $\mathcal{W}R = \{ (S(\mathbf{i}) \to T(\mathbf{j})) : f(\mathbf{i},\mathbf{j}) \}$

Wrapping, Unwrapping, Domain Map and Range Map

$$R = \{ S(\mathbf{i}) \to T(\mathbf{j}) : f(\mathbf{i},\mathbf{j}) \} \qquad S = \{ (U(\mathbf{i}) \to V(\mathbf{j})) : g(\mathbf{i},\mathbf{j}) \}$$

- Wrap (`iscc: wrap`)

  $\mathcal{W}R = \{ (S(\mathbf{i}) \to T(\mathbf{j})) : f(\mathbf{i},\mathbf{j}) \}$
- Unwrap (`iscc: unwrap`)

  $\mathcal{W}^{-1}S = \{ U(\mathbf{i}) \to V(\mathbf{j}) : g(\mathbf{i},\mathbf{j}) \}$

Wrapping, Unwrapping, Domain Map and Range Map

$$R = \{ S(\mathbf{i}) \to T(\mathbf{j}) : f(\mathbf{i},\mathbf{j}) \} \qquad S = \{ (U(\mathbf{i}) \to V(\mathbf{j})) : g(\mathbf{i},\mathbf{j}) \}$$

- Wrap (iscc: `wrap`)

  $\mathcal{W}R = \{ (S(\mathbf{i}) \to T(\mathbf{j})) : f(\mathbf{i},\mathbf{j}) \}$
- Unwrap (iscc: `unwrap`)

  $\mathcal{W}^{-1}S = \{ U(\mathbf{i}) \to V(\mathbf{j}) : g(\mathbf{i},\mathbf{j}) \}$
- Domain map (iscc: `domain_map`)

  $\underrightarrow{\mathrm{dom}}\, R = \{ (S(\mathbf{i}) \to T(\mathbf{j})) \to S(\mathbf{i}) : f(\mathbf{i},\mathbf{j}) \}$

  $I = \{ (\mathtt{S}(i) \to \mathtt{R1}()) : 0 \le i < n \}$

  $\underrightarrow{\mathrm{dom}}(\mathcal{W}^{-1}I) = \{ (\mathtt{S}(i) \to \mathtt{R1}()) \to \mathtt{S}(i) : 0 \le i < n \}$

  $\Rightarrow$ maps tagged instance set to untagged instance set

  $\Rightarrow$ precompose with instance set based relations to obtain tagged instance set based relations

Wrapping, Unwrapping, Domain Map and Range Map

$$R = \{ S(\mathbf{i}) \to T(\mathbf{j}) : f(\mathbf{i},\mathbf{j}) \} \qquad S = \{ (U(\mathbf{i}) \to V(\mathbf{j})) : g(\mathbf{i},\mathbf{j}) \}$$

- Wrap (iscc: `wrap`)

  $\mathcal{W}R = \{ (S(\mathbf{i}) \to T(\mathbf{j})) : f(\mathbf{i},\mathbf{j}) \}$
- Unwrap (iscc: `unwrap`)

  $\mathcal{W}^{-1}S = \{ U(\mathbf{i}) \to V(\mathbf{j}) : g(\mathbf{i},\mathbf{j}) \}$
- Domain map (iscc: `domain_map`)

  $\underrightarrow{\mathrm{dom}}\, R = \{ (S(\mathbf{i}) \to T(\mathbf{j})) \to S(\mathbf{i}) : f(\mathbf{i},\mathbf{j}) \}$

  $I = \{ (\mathrm{S}(i) \to \mathtt{R1}()) : 0 \le i < n \}$

  $\underrightarrow{\mathrm{dom}}(\mathcal{W}^{-1}I) = \{ (\mathrm{S}(i) \to \mathtt{R1}()) \to \mathrm{S}(i) : 0 \le i < n \}$

  $\Rightarrow$ maps tagged instance set to untagged instance set

  $\Rightarrow$ precompose with instance set based relations to obtain tagged instance set based relations
- Range map (iscc: `range_map`)

  $\underrightarrow{\mathrm{ran}}\, R = \{ (S(\mathbf{i}) \to T(\mathbf{j})) \to T(\mathbf{j}) : f(\mathbf{i},\mathbf{j}) \}$

Parametric Example: Matrix Multiplication

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:       C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
```

- Tagged Access Relations

  $W = \{ (\mathtt{S1}(i,j) \to \mathtt{R0}()) \to \mathtt{C}(i,j); (\mathtt{S2}(i,j,k) \to \mathtt{R1}()) \to \mathtt{C}(i,j) \}$

  $R = \{ (\mathtt{S2}(i,j,k) \to \mathtt{R2}()) \to \mathtt{C}(i,j); (\mathtt{S2}(i,j,k) \to \mathtt{R3}()) \to \mathtt{A}(i,k);$
  $(\mathtt{S2}(i,j,k) \to \mathtt{R4}()) \to \mathtt{B}(k,j) \}$

- Tagged Schedule

  $\{ (\mathtt{S1}(i,j) \to \mathtt{R0}()) \to (i,j,0,0); (\mathtt{S2}(i,j,k) \to \mathtt{R1}()) \to (i,j,1,k)$
  $(\mathtt{S2}(i,j,k) \to \mathtt{R2}()) \to (i,j,1,k); (\mathtt{S2}(i,j,k) \to \mathtt{R3}()) \to (i,j,1,k);$
  $(\mathtt{S2}(i,j,k) \to \mathtt{R4}()) \to (i,j,1,k) \}$

- Tagged Dataflow

  $\{ (\mathtt{S1}(i,j) \to \mathtt{R0}()) \to (\mathtt{S2}(i,j,0) \to \mathtt{R2}())$
  $(\mathtt{S2}(i,j,k) \to \mathtt{R1}()) \to (\mathtt{S2}(i,j,k+1) \to \mathtt{R2}()) \}$

Maximal Static Expansion

[3]

```
for (int i = 0; i < n; ++i) {
S1:       t = f1(i);
S2:       A[i] = t;
S3:       t = f2(i);
S4:       if (f3(i))
S5:             t = f4(i);
S6:       B[i] = t;
}
```

Dataflow cannot be determined independently of run-time information

Maximal Static Expansion

[3]

```
for (int i = 0; i < n; ++i) {
S1:       t = f1(i);
S2:       A[i] = t;
S3:       t = f2(i);
S4:       if (f3(i))
S5:             t = f4(i);
S6:       B[i] = t;
}
```

Dataflow cannot be determined independently of run-time information

$\Rightarrow$ approximate dataflow

$\{ \mathtt{S1}(i) \rightarrow \mathtt{S2}(i); \mathtt{S3}(i) \rightarrow \mathtt{S6}(i); \mathtt{S5}(i) \rightarrow \mathtt{S6}(i) \}$

Maximal Static Expansion

[3]

```
for (int i = 0; i < n; ++i) {
S1:       t = f1(i);
S2:       A[i] = t;
S3:       t = f2(i);
S4:       if (f3(i))
S5:             t = f4(i);
S6:       B[i] = t;
}
```

Dataflow cannot be determined independently of run-time information

$\Rightarrow$ approximate dataflow

$\{ \mathtt{S1}(i) \rightarrow \mathtt{S2}(i); \mathtt{S3}(i) \rightarrow \mathtt{S6}(i); \mathtt{S5}(i) \rightarrow \mathtt{S6}(i) \}$

$\Rightarrow$ a read may be associated to more than one write

$\Rightarrow$ corresponding equivalence classes should not be expanded apart

Maximal Static Expansion

[3]

```
for (int i = 0; i < n; ++i) {
S1:       t = f1(i);                    t1[i] = f1(i);
S2:       A[i] = t;                     A[i] = t1[i];
S3:       t = f2(i);                    t2[i] = f2(i);
S4:       if (f3(i))                    if (f3(i))
S5:           t = f4(i);                    t2[i] = f4(i);
S6:       B[i] = t;                     B[i] = t2[i];
}
```

Dataflow cannot be determined independently of run-time information

$\Rightarrow$ approximate dataflow

$\{ \mathrm{S1}(i) \rightarrow \mathrm{S2}(i); \mathrm{S3}(i) \rightarrow \mathrm{S6}(i); \mathrm{S5}(i) \rightarrow \mathrm{S6}(i) \}$

$\Rightarrow$ a read may be associated to more than one write

$\Rightarrow$ corresponding equivalence classes should not be expanded apart

Approximate Dataflow Analysis

How to compute dataflow in presence of data dependent control?

Two approaches

- Direct computation
  - distinguish between may- and must-writes

Approximate Dataflow Analysis

How to compute dataflow in presence of data dependent control?

Two approaches

- Direct computation
  - distinguish between may- and must-writes
- Derived from exact run-time dependent dataflow
  - compute exact dataflow in terms of run-time information
  - exploit properties of run-time information
  - project out run-time information

Approximate Dataflow Analysis

How to compute dataflow in presence of data dependent control?

Two approaches

- Direct computation
  - distinguish between may- and must-writes
- Derived from exact run-time dependent dataflow
  - compute exact dataflow in terms of run-time information
  - exploit properties of run-time information
  - project out run-time information

May Writes

Keep track of whether write is possible or definite

- Must-writes
  Array elements that are definitely accessed by statement instance
- May-writes
  Array elements that are possibly accessed by statement instance

Must-write access relation is subset of may-write access relation

May Writes

Keep track of whether write is possible or definite

- Must-writes
  Array elements that are definitely accessed by statement instance
- May-writes
  Array elements that are possibly accessed by statement instance
  - statement instance not necessarily executed

```
for (i = 0; i < n; ++i)
    if (A[i] > 0)
S:      B[i] = A[i];
```

May-write: $\{ \mathtt{S}(i) \rightarrow \mathtt{B}(i) \}$

Must-write access relation is subset of may-write access relation

May Writes

Keep track of whether write is possible or definite

- Must-writes
  Array elements that are definitely accessed by statement instance
- May-writes
  Array elements that are possibly accessed by statement instance
  - statement instance not necessarily executed

```
for (i = 0; i < n; ++i)
    if (A[i] > 0)
S:      B[i] = A[i];
```

    May-write: $\{ \mathtt{S}(i) \rightarrow \mathtt{B}(i) \}$
  - array element not necessarily accessed

```
int A[N];
/* ... */
T:  A[B[0]] = 5;
```

    May-write: $\{ \mathtt{T}() \rightarrow \mathtt{A}(a) : 0 \leq a < N \}$

Must-write access relation is subset of may-write access relation

Approximate Dataflow — Direct Computation

- Read after write dependences
  - write and read access same memory location
  - write executed before the read

Approximate Dataflow — Direct Computation

- Read after write dependences
  - write and read access same memory location
  - write executed before the read

- Dataflow dependences
  - write and read access same memory location
  - write executed before the read
  - no intermediate write to same memory location
    $\Rightarrow$ intermediate write kills dependence

Approximate Dataflow — Direct Computation

- Read after write dependences
  - write and read access same memory location
  - write executed before the read
- Dataflow dependences
  - write and read access same memory location
  - write executed before the read
  - no intermediate write to same memory location
    $\Rightarrow$ intermediate write kills dependence
- Approximate dataflow dependences
  - may-write and read access same memory location
  - may-write executed before the read
  - no intermediate must-write to same memory location
    $\Rightarrow$ intermediate must-write kills dependence

Approximate Dataflow — Direct Computation

- Read after write dependences
  - write and read access same memory location
  - write executed before the read

  $\Rightarrow$ Approximate dataflow analysis with no must-writes
- Dataflow dependences
  - write and read access same memory location
  - write executed before the read
  - no intermediate write to same memory location

    $\Rightarrow$ intermediate write kills dependence
- Approximate dataflow dependences
  - may-write and read access same memory location
  - may-write executed before the read
  - no intermediate must-write to same memory location

    $\Rightarrow$ intermediate must-write kills dependence

Performing Dataflow Analysis

[9, 17]

Two possibilities

1. dataflow analysis on polyhedral model
   - first extract instance set, access relations and schedule
   - then perform dataflow analysis

   E.g., `isl`
2. dataflow analysis on AST before/during model extraction
   Proposed by, e.g., Maslov (1994)

Performing Dataflow Analysis

[9, 17]

Two possibilities

1. dataflow analysis on polyhedral model
   - first extract instance set, access relations and schedule
   - then perform dataflow analysis

   E.g., `isl`
2. dataflow analysis on AST before/during model extraction
   Proposed by, e.g., Maslov (1994)

Dataflow in Parallel Programs

1. use refined execution order during dataflow analysis
2. remove spurious dependences after dataflow analysis

   Note: dataflow from must-writes cannot be removed without caution

   $\Rightarrow$ must-write may have killed other dependences
   $\Rightarrow$ other dependences may have to be added back

Explicit Kills

```
     int A[N];

S:   A[0] = 1;

     for (int i = 0; i < N; ++i)
T:       A[perm[i]] = f(i);
U:   f(A[0]);
```

- Assume `perm` represents a permutation

  $\Rightarrow$ there can be no dataflow from `S` to `U`

Explicit Kills

```
     int A[N];

S:   A[0] = 1;

     for (int i = 0; i < N; ++i)
T:       A[perm[i]] = f(i);
U:   f(A[0]);
```

- Assume `perm` represents a permutation
  - $\Rightarrow$ there can be no dataflow from `S` to `U`
- Compiler does not know all elements are written by `T`
  - $\Rightarrow$ may find dataflow from `S` to `U`
  - $\Rightarrow$ user can insert explicit kill
  - $\Rightarrow$ explicit kill used to kill dependences, just like must-write

Explicit Kills

```
     int A[N];

S:   A[0] = 1;
     __pencil_kill(A);
     for (int i = 0; i < N; ++i)
T:       A[perm[i]] = f(i);
U:   f(A[0]);
```

- Assume `perm` represents a permutation
  - $\Rightarrow$ there can be no dataflow from `S` to `U`
- Compiler does not know all elements are written by `T`
  - $\Rightarrow$ may find dataflow from `S` to `U`
  - $\Rightarrow$ user can insert explicit kill
  - $\Rightarrow$ explicit kill used to kill dependences, just like must-write

Local Variables and Kills

```
for (int i = 0; i < N; ++i) {
    int t;
    /* ... */
}
```

⇒ there can be no dataflow on `t` across different iterations

⇒ pet automatically inserts kills

- before declaration and
- at end of block containing declaration

Killing False Dependences

Dataflow derived from read after write dependences through killing.

Should we do the same for false dependences?

Killing False Dependences

Dataflow derived from read after write dependences through killing.

Should we do the same for false dependences?

$\Rightarrow$ no need for validity schedule constraints

$\Rightarrow$ removed dependences implied by remaining dependences

Killing False Dependences

Dataflow derived from read after write dependences through killing.

Should we do the same for false dependences?

$\Rightarrow$ no need for validity schedule constraints

$\Rightarrow$ removed dependences implied by remaining dependences

Optimizing locality

- typical criterion: minimize maximal dependence distance

  dependence distances: $\{ P(S(\mathbf{i})) - P(T(\mathbf{j})) : S(\mathbf{i}) \to T(\mathbf{j}) \in D \}$

  D: (local) dependence relation, P: partial schedule

Killing False Dependences

Dataflow derived from read after write dependences through killing.

Should we do the same for false dependences?

$\Rightarrow$ no need for validity schedule constraints

$\Rightarrow$ removed dependences implied by remaining dependences

Optimizing locality

- typical criterion: minimize maximal dependence distance

  dependence distances: $\{ P(S(\mathbf{i})) - P(T(\mathbf{j})) : S(\mathbf{i}) \to T(\mathbf{j}) \in D \}$

  D: (local) dependence relation, P: partial schedule
- may also be useful for false dependences if no expansion is performed

  $\Rightarrow$ locality of reused memory location
- killing false dependences avoids critical path determined by transitively covered dependences

  $\Rightarrow$ allow must-writes to kill dependences, but not explicit kills

Approximate Dataflow Analysis

How to compute dataflow in presence of data dependent control?

Two approaches

- Direct computation
  - distinguish between may- and must-writes
- Derived from exact run-time dependent dataflow
  - compute exact dataflow in terms of run-time information
  - exploit properties of run-time information
  - project out run-time information

Approximate Dataflow Analysis

How to compute dataflow in presence of data dependent control?

Two approaches

- Direct computation
  - distinguish between may- and must-writes
- Derived from exact run-time dependent dataflow
  - compute exact dataflow in terms of run-time information
  - exploit properties of run-time information
  - project out run-time information

Run-time Dependent Dataflow Analysis

[4, 25]

Approaches

- “fuzzy array dataflow analysis”
- “on-demand-parametric array dataflow analysis”

Run-time Dependent Dataflow Analysis

[4, 25]

Approaches

- "fuzzy array dataflow analysis"
- "on-demand-parametric array dataflow analysis"

Run-time Dependent Dataflow Analysis

[4, 25]

Approaches

- "fuzzy array dataflow analysis"
- "on-demand-parametric array dataflow analysis"

```
for (int i = 0; i < n; ++i) {
S1:       t = f1(i);
S2:       A[i] = t;
S3:       t = f2(i);
S4:       if (f3(i))
S5:             t = f4(i);
S6:       B[i] = t;
}
```

Run-time Dependent Dataflow Analysis

[4, 25]

Approaches

- "fuzzy array dataflow analysis"
- "on-demand-parametric array dataflow analysis"

```
for (int i = 0; i < n; ++i) {
S1:      t = f1(i);
S2:      A[i] = t;
S3:      t = f2(i);
S4:      if (f3(i))
S5:          t = f4(i);
S6:      B[i] = t;
}
```

- Run-time dependent dataflow

  $\{ \mathtt{S1}(i) \rightarrow \mathtt{S2}(i); \mathtt{S3}(i) \rightarrow \mathtt{S6}(i) : \beta^{\mathtt{S5}}_{\mathtt{S6}} = 0; \mathtt{S5}(i) \rightarrow \mathtt{S6}(i) : \beta^{\mathtt{S5}}_{\mathtt{S6}} = 1 \}$

  β^P_C: any potential source instance P is executed for sink C

  $\boldsymbol{\lambda}^P_C$: last potential source instance P executed for sink C

Run-time Dependent Dataflow Analysis

[4, 25]

Approaches

- "fuzzy array dataflow analysis"
- "on-demand-parametric array dataflow analysis"

```
for (int i = 0; i < n; ++i) {
S1:     t = f1(i);
S2:     A[i] = t;
S3:     t = f2(i);
S4:     if (f3(i))
S5:         t = f4(i);
S6:     B[i] = t;
}
```

- Run-time dependent dataflow

  $\{\, \mathtt{S1}(i) \rightarrow \mathtt{S2}(i); \mathtt{S3}(i) \rightarrow \mathtt{S6}(i) : \beta^{\mathtt{S5}}_{\mathtt{S6}} = 0; \mathtt{S5}(i) \rightarrow \mathtt{S6}(i) : \beta^{\mathtt{S5}}_{\mathtt{S6}} = 1 \,\}$

  β^P_C: any potential source instance P is executed for sink C

  $\boldsymbol{\lambda}^P_C$: last potential source instance P executed for sink C
- Approximate dataflow (project out β and $\boldsymbol{\lambda}$)

  $\{\, \mathtt{S1}(i) \rightarrow \mathtt{S2}(i); \mathtt{S3}(i) \rightarrow \mathtt{S6}(i); \mathtt{S5}(i) \rightarrow \mathtt{S6}(i) \,\}$

Representing Dynamic Conditions

```
N1:  n = f();
     for (int k = 0; k < 100; ++k) {
M:       m = g();
         for (int i = 0; i < m; ++i)
             for (int j = 0; j < n; ++j)
A:               a[j][i] = g();
N2:      n = f();
     }
```

What is instance set (restricted to A statement)?

Representing Dynamic Conditions

```
N1:  n = f();
     for (int k = 0; k < 100; ++k) {
M:       m = g();
         for (int i = 0; i < m; ++i)
             for (int j = 0; j < n; ++j)
A:               a[j][i] = g();
N2:      n = f();
     }
```

What is instance set (restricted to A statement)?

$\{ \mathtt{A}(k, i, j) : 0 \leq k < 100 \wedge 0 \leq i < \mathtt{m} \wedge 0 \leq j < \mathtt{n} \}$?

Representing Dynamic Conditions

```
N1:  n = f();
     for (int k = 0; k < 100; ++k) {
M:       m = g();
         for (int i = 0; i < m; ++i)
             for (int j = 0; j < n; ++j)
A:               a[j][i] = g();
N2:      n = f();
     }
```

What is instance set (restricted to A statement)?

$\{ \mathtt{A}(k,i,j) : 0 \leq k < 100 \wedge 0 \leq i < \mathtt{m} \wedge 0 \leq j < \mathtt{n} \}$?

$\Rightarrow$ no, m and n cannot be treated as symbolic constants (they are modified inside k-loop)

Representing Dynamic Conditions

```
N1:  n = f();
     for (int k = 0; k < 100; ++k) {
M:       m = g();
         for (int i = 0; i < m; ++i)
             for (int j = 0; j < n; ++j)
A:               a[j][i] = g();
N2:      n = f();
     }
```

What is instance set (restricted to A statement)?

$\{ \mathtt{A}(k,i,j) : 0 \leq k < 100 \wedge 0 \leq i < \mathtt{m} \wedge 0 \leq j < \mathtt{n} \}$?

$\Rightarrow$ no, m and n cannot be treated as symbolic constants
(they are modified inside k-loop)

$\{\mathtt{A}(k,i,j) : 0 \leq k < 100 \wedge 0 \leq i < \mathtt{valueOf\_m}(k) \wedge 0 \leq j < \mathtt{valueOf\_n}(k)\}$?

Representing Dynamic Conditions

```
N1:  n = f();
     for (int k = 0; k < 100; ++k) {
M:       m = g();
         for (int i = 0; i < m; ++i)
             for (int j = 0; j < n; ++j)
A:               a[j][i] = g();
N2:      n = f();
     }
```

What is instance set (restricted to A statement)?

$\{ \mathtt{A}(k,i,j) : 0 \leq k < 100 \wedge 0 \leq i < \mathtt{m} \wedge 0 \leq j < \mathtt{n} \}$?

$\Rightarrow$ no, m and n cannot be treated as symbolic constants
(they are modified inside k-loop)

$\{\mathtt{A}(k,i,j) : 0 \leq k < 100 \wedge 0 \leq i < \mathtt{valueOf\_m}(k) \wedge 0 \leq j < \mathtt{valueOf\_n}(k)\}$?

$\Rightarrow$ requires uninterpreted functions (of arity > 0)

Representing Dynamic Conditions

```
N1:  n = f();
     for (int k = 0; k < 100; ++k) {
M:       m = g();
         for (int i = 0; i < m; ++i)
             for (int j = 0; j < n; ++j)
A:               a[j][i] = g();
N2:      n = f();
     }
```

What is instance set (restricted to A statement)?

$\{ \mathtt{A}(k,i,j) : 0 \leq k < 100 \wedge 0 \leq i < \mathtt{m} \wedge 0 \leq j < \mathtt{n} \}$?

$\Rightarrow$ no, m and n cannot be treated as symbolic constants
(they are modified inside k-loop)

$\{\mathtt{A}(k,i,j) : 0 \leq k < 100 \wedge 0 \leq i < \mathtt{valueOf\_m}(k) \wedge 0 \leq j < \mathtt{valueOf\_n}(k)\}$?

$\Rightarrow$ requires uninterpreted functions (of arity > 0)

Alternative: use overapproximation of instance set and keep track of which elements are executed

Representing Dynamic Conditions

```
N1:  n = f();
     for (int k = 0; k < 100; ++k) {
M:       m = g();
         for (int i = 0; i < m; ++i)
             for (int j = 0; j < n; ++j)
A:               a[j][i] = g();
N2:      n = f();
     }
```

- Instance set: $\{ \mathtt{A}(k,i,j) : 0 \leq k < 100 \wedge 0 \leq i \wedge 0 \leq j \}$
- Filter:
  - Filter access relations: reader $\rightarrow$ (writer $\rightarrow$ array element)
    - $F_1^{\mathtt{A}} = \{ \mathtt{A}(k,i,j) \rightarrow (\mathtt{M}(k) \rightarrow \mathtt{m}()) \}$
    - $F_2^{\mathtt{A}} = \{ \mathtt{A}(0,i,j) \rightarrow (\mathtt{N1}() \rightarrow \mathtt{n}()); \mathtt{A}(k,i,j) \rightarrow (\mathtt{N2}(k-1) \rightarrow \mathtt{n}()) : k \geq 1 \}$
  - Filter value relation:
    $V^{\mathtt{A}} = \{ \mathtt{A}(k,i,j) \rightarrow (m,n) : 0 \leq k \leq 99 \wedge 0 \leq i < m \wedge 0 \leq j < n \}$

Statement instance is executed iff values written by corresponding write accesses (through filter access relations) satisfy filter value relation

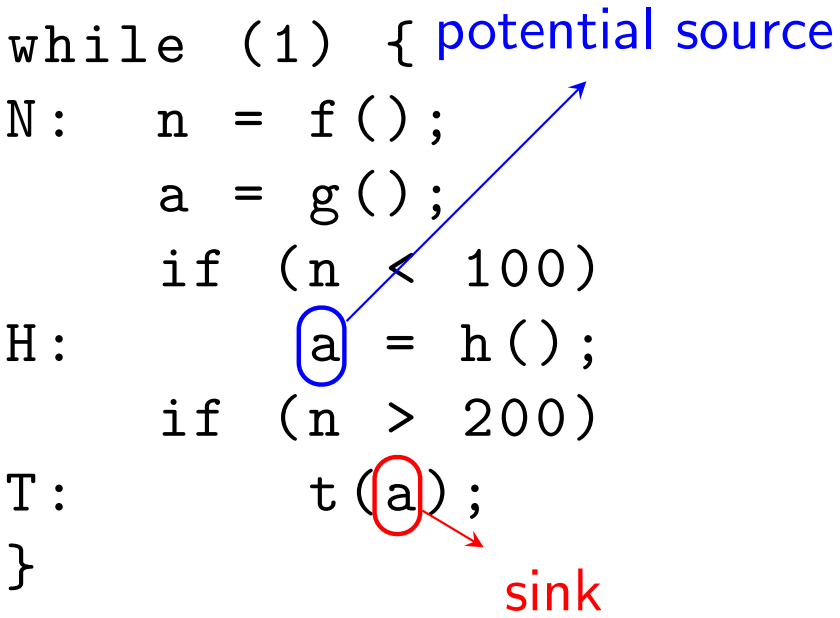
Representing Dynamic Conditions

```
N1:  n = f();
     for (int k = 0; k < 100; ++k) {
M:       m = g();
         for (int i = 0; i < m; ++i)
             for (int j = 0; j < n; ++j)
A:               a[j][i] = g();
N2:      n = f();
     }
```

- Instance set: $\{ \mathtt{A}(k,i,j) : 0 \leq k < 100 \wedge 0 \leq i \wedge 0 \leq j \}$
- Filter:
  - Filter access relations: reader $\rightarrow$ (writer $\rightarrow$ array element)
    - $F_1^{\mathtt{A}} = \{ \mathtt{A}(k,i,j) \rightarrow (\mathtt{M}(k) \rightarrow \mathtt{m}()) \}$
    - $F_2^{\mathtt{A}} = \{ \mathtt{A}(0,i,j) \rightarrow (\mathtt{N1}() \rightarrow \mathtt{n}()); \mathtt{A}(k,i,j) \rightarrow (\mathtt{N2}(k-1) \rightarrow \mathtt{n}()) : k \geq 1 \}$
  - Filter value relation:
    $V^{\mathtt{A}} = \{ \mathtt{A}(k,i,j) \rightarrow (m,n) : 0 \leq k \leq 99 \wedge 0 \leq i < m \wedge 0 \leq j < n \}$

Statement instance is executed iff values written by corresponding write accesses (through filter access relations) satisfy filter value relation

Parametric Array Dataflow Analysis



Is there any dataflow between potential source and sink at inner level?

Parametric Array Dataflow Analysis

```
while (1) {
N:   n = f();
     a = g();
     if (n < 100)
H:       a = h();
     if (n > 200)
T:       t(a);
}
```

potential source

sink

$$I = \{\mathtt{H}(i) : i \geq 0; \mathtt{T}(i) : i \geq 0\}$$
$$F^{\mathtt{H}} = \{\mathtt{H}(i) \rightarrow (\mathtt{N}(i) \rightarrow \mathtt{n}())\}$$
$$V^{\mathtt{H}} = \{\mathtt{H}(i) \rightarrow (n) : i \geq 0 \wedge n < 100\}$$
$$F^{\mathtt{T}} = \{\mathtt{T}(i) \rightarrow (\mathtt{N}(i) \rightarrow \mathtt{n}())\}$$
$$V^{\mathtt{T}} = \{\mathtt{T}(i) \rightarrow (n) : i \geq 0 \wedge n > 200\}$$

Is there any dataflow between potential source and sink at inner level?

Parametric Array Dataflow Analysis

```
while (1) {
N:   n = f();
     a = g();
     if (n < 100)
H:       a = h();
     if (n > 200)
T:       t(a);
}
```

potential source

sink

$$I = \{ \mathtt{H}(i) : i \geq 0; \mathtt{T}(i) : i \geq 0 \}$$
$$F^{\mathtt{H}} = \{ \mathtt{H}(i) \rightarrow (\mathtt{N}(i) \rightarrow \mathtt{n}()) \}$$
$$V^{\mathtt{H}} = \{ \mathtt{H}(i) \rightarrow (n) : i \geq 0 \wedge n < 100 \}$$
$$F^{\mathtt{T}} = \{ \mathtt{T}(i) \rightarrow (\mathtt{N}(i) \rightarrow \mathtt{n}()) \}$$
$$V^{\mathtt{T}} = \{ \mathtt{T}(i) \rightarrow (n) : i \geq 0 \wedge n > 200 \}$$

Is there any dataflow between potential source and sink at inner level?

- $M = \{ \mathtt{T}(i) \rightarrow \mathtt{H}(i) \}$

Parametric Array Dataflow Analysis

```
while (1) {
N:   n = f();
     a = g();
     if (n < 100)
H:       a = h();
     if (n > 200)
T:       t(a);
}
```

potential source

sink

$$I = \{ \mathtt{H}(i) : i \geq 0; \mathtt{T}(i) : i \geq 0 \}$$
$$F^{\mathtt{H}} = \{ \mathtt{H}(i) \rightarrow (\mathtt{N}(i) \rightarrow \mathtt{n}()) \}$$
$$V^{\mathtt{H}} = \{ \mathtt{H}(i) \rightarrow (n) : i \geq 0 \wedge n < 100 \}$$
$$F^{\mathtt{T}} = \{ \mathtt{T}(i) \rightarrow (\mathtt{N}(i) \rightarrow \mathtt{n}()) \}$$
$$V^{\mathtt{T}} = \{ \mathtt{T}(i) \rightarrow (n) : i \geq 0 \wedge n > 200 \}$$

Is there any dataflow between potential source and sink at inner level?

- $M = \{ \mathtt{T}(i) \rightarrow \mathtt{H}(i) \}$
- $F^{\mathtt{H}} \circ M \subseteq F^{\mathtt{T}}$

  $\Rightarrow$ filter elements accessed by any potential source instance associated to sink instance forms subset of filter elements accessed by sink instance

Parametric Array Dataflow Analysis

```
while (1) {
N:   n = f();
     a = g();
     if (n < 100)
H:       a = h();
     if (n > 200)
T:       t(a);
}
```

potential source

sink

$$I = \{ \mathtt{H}(i) : i \geq 0; \mathtt{T}(i) : i \geq 0 \}$$
$$F^{\mathtt{H}} = \{ \mathtt{H}(i) \to (\mathtt{N}(i) \to \mathtt{n}()) \}$$
$$V^{\mathtt{H}} = \{ \mathtt{H}(i) \to (n) : i \geq 0 \wedge n < 100 \}$$
$$F^{\mathtt{T}} = \{ \mathtt{T}(i) \to (\mathtt{N}(i) \to \mathtt{n}()) \}$$
$$V^{\mathtt{T}} = \{ \mathtt{T}(i) \to (n) : i \geq 0 \wedge n > 200 \}$$

Is there any dataflow between potential source and sink at inner level?

- $M = \{ \mathtt{T}(i) \to \mathtt{H}(i) \}$
- $F^{\mathtt{H}} \circ M \subseteq F^{\mathtt{T}}$
  - $\Rightarrow$ filter elements accessed by any potential source instance associated to sink instance forms subset of filter elements accessed by sink instance
  - $\Rightarrow$ constraints on filter values at sink also apply at corresponding potential source: $V^{\mathtt{T}} \circ M^{-1} = \{ \mathtt{H}(i) \to (n) : i \geq 0 \wedge n > 200 \}$

Parametric Array Dataflow Analysis

```
while (1) {
N:   n = f();
     a = g();
     if (n < 100)
H:       a = h();
     if (n > 200)
T:       t(a);
}
```

potential source

sink

$$I = \{ \mathtt{H}(i) : i \geq 0; \mathtt{T}(i) : i \geq 0 \}$$
$$F^{\mathtt{H}} = \{ \mathtt{H}(i) \to (\mathtt{N}(i) \to \mathtt{n}()) \}$$
$$V^{\mathtt{H}} = \{ \mathtt{H}(i) \to (n) : i \geq 0 \wedge n < 100 \}$$
$$F^{\mathtt{T}} = \{ \mathtt{T}(i) \to (\mathtt{N}(i) \to \mathtt{n}()) \}$$
$$V^{\mathtt{T}} = \{ \mathtt{T}(i) \to (n) : i \geq 0 \wedge n > 200 \}$$

Is there any dataflow between potential source and sink at inner level?

- $M = \{ \mathtt{T}(i) \to \mathtt{H}(i) \}$
- $F^{\mathtt{H}} \circ M \subseteq F^{\mathtt{T}}$
  - $\Rightarrow$ filter elements accessed by any potential source instance associated to sink instance forms subset of filter elements accessed by sink instance
  - $\Rightarrow$ constraints on filter values at sink also apply at corresponding potential source: $V^{\mathtt{T}} \circ M^{-1} = \{ \mathtt{H}(i) \to (n) : i \geq 0 \wedge n > 200 \}$
- $\left(V^{\mathtt{T}} \circ M^{-1}\right) \cap V^{\mathtt{H}} = \emptyset$
  - $\Rightarrow$ there can be no dataflow at inner level

Polyhedral Process Networks

[19]

- Main purpose: extract task level parallelism from dataflow graph

$$\begin{aligned} \text{statement} &\rightarrow \text{process} \\ \text{flow dependence} &\rightarrow \text{communication channel} \end{aligned}$$

  $\Rightarrow$ requires dataflow analysis

- Processes are mapped to parallel hardware (e.g., FPGA)

Polyhedral Process Networks

[19]

- Main purpose: extract task level parallelism from dataflow graph

$$\begin{aligned} \text{statement} &\rightarrow \text{process} \\ \text{flow dependence} &\rightarrow \text{communication channel} \end{aligned}$$

  $\Rightarrow$ requires dataflow analysis
- Processes are mapped to parallel hardware (e.g., FPGA)

Example:

```
for (int i = 0; i < n; ++i) {
S:      t = f1(A[i]);
T:      B[i] = f2(t);
}
```

Polyhedral Process Networks

[19]

- Main purpose: extract task level parallelism from dataflow graph

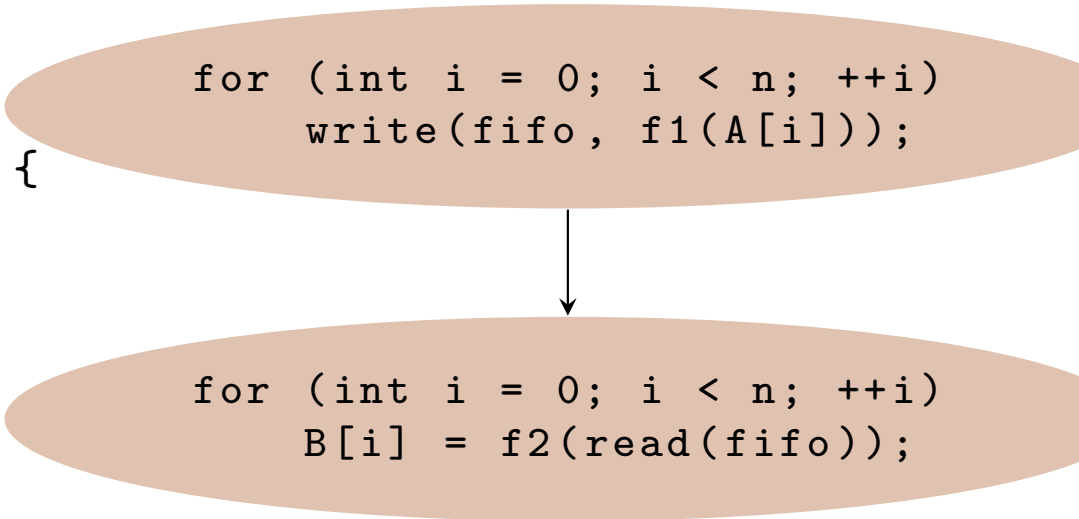
$$\begin{aligned} \text{statement} &\rightarrow \text{process} \\ \text{flow dependence} &\rightarrow \text{communication channel} \end{aligned}$$

$\Rightarrow$ requires dataflow analysis

- Processes are mapped to parallel hardware (e.g., FPGA)

Example:

```
for (int i = 0; i < n; ++i) {
S:      t = f1(A[i]);
T:      B[i] = f2(t);
}
```



Process Networks with Dynamic Control

```
for (int i = 0; i < n; ++i) {
S1:       t = f1(i);
S2:       A[i] = t;
S3:       t = f2(i);
S4:       if (f3(i))
S5:                 t = f4(i);
S6:       B[i] = t;
}
```

Run-time dependent dataflow:

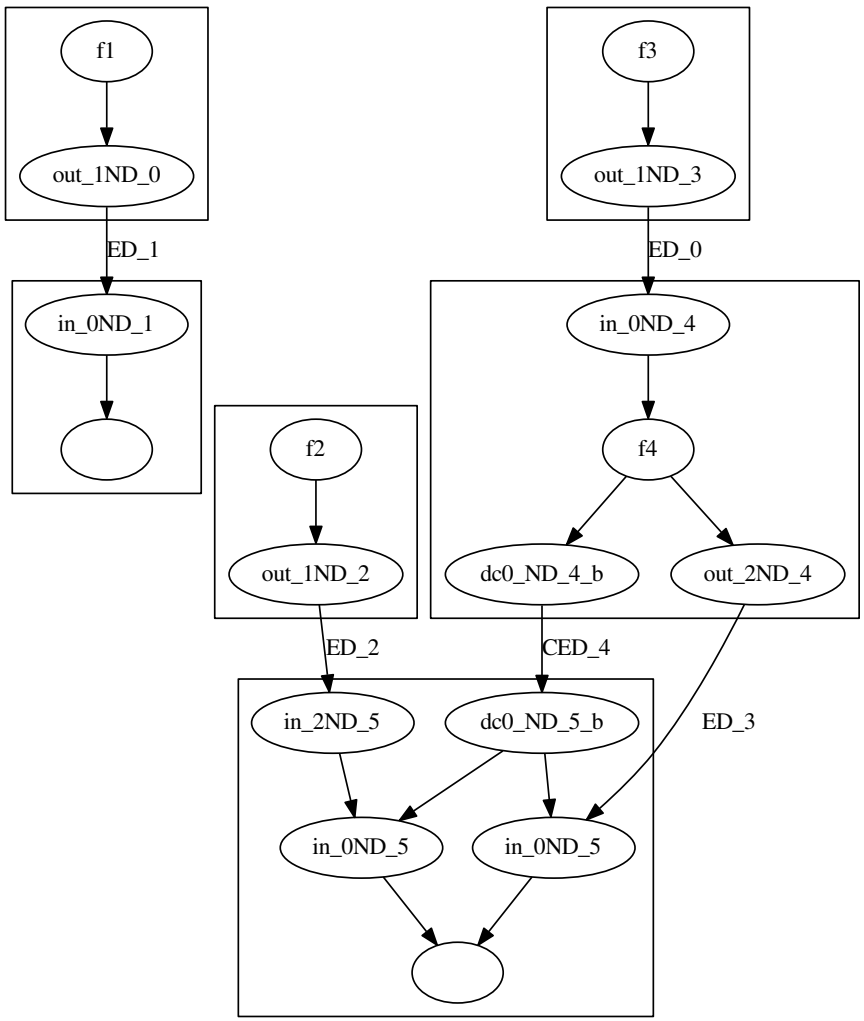
$$\{\, \mathrm{S1}(i) \to \mathrm{S2}(i); \mathrm{S3}(i) \to \mathrm{S6}(i) : \beta^{\mathrm{S5}}_{\mathrm{S6}} = 0;$$
$$\mathrm{S5}(i) \to \mathrm{S6}(i) : \beta^{\mathrm{S5}}_{\mathrm{S6}} = 1; \mathrm{S4}(i) \to \mathrm{S5}(i) \,\}$$

Process Networks with Dynamic Control

```
for (int i = 0; i < n; ++i) {
S1:       t = f1(i);
S2:       A[i] = t;
S3:       t = f2(i);
S4:       if (f3(i))
S5:             t = f4(i);
S6:       B[i] = t;
}
```

Run-time dependent dataflow:

$$\{\, \mathtt{S1}(i) \to \mathtt{S2}(i); \mathtt{S3}(i) \to \mathtt{S6}(i) : \beta^{\mathtt{S5}}_{\mathtt{S6}} = 0;$$
$$\mathtt{S5}(i) \to \mathtt{S6}(i) : \beta^{\mathtt{S5}}_{\mathtt{S6}} = 1; \mathtt{S4}(i) \to \mathtt{S5}(i) \,\}$$



Reductions

```
A:   s = 0;
     for (int i = 0; i < n; ++i)
B:       s += A[i];
C:   B[0] = s;
```

Dataflow:

$\{ \mathtt{A}() \to \mathtt{B}(0); \mathtt{B}(i) \to \mathtt{B}(i+1) : 0 \leq i < n-1; \mathtt{B}(n-1) \to \mathtt{C}() \}$

$\Rightarrow$ fixes order of reduction

Reductions

```
A:   s = 0;
     for (int i = 0; i < n; ++i)
B:       s += A[i];
C:   B[0] = s;
```

Dataflow:

$\{ \mathtt{A}() \to \mathtt{B}(0); \mathtt{B}(i) \to \mathtt{B}(i+1) : 0 \leq i < n-1; \mathtt{B}(n-1) \to \mathtt{C}() \}$

$\Rightarrow$ fixes order of reduction

Allow reordering of updates:

- read in C should depend on all updates in B
  - $\Rightarrow$ updates should not kill dependences
  - $\Rightarrow$ remove updates from must-writes
- updates should not depend on each other
  - $\Rightarrow$ remove false dependences between updates that flow to the same read, provided the read does not also depend on intermediate writes

Reductions

```
A:   s = 0;
     for (int i = 0; i < n; ++i)
B:       s += A[i];
C:   B[0] = s;
```

Dataflow:

$\{ \mathtt{A}() \to \mathtt{B}(0); \mathtt{B}(i) \to \mathtt{B}(i+1) : 0 \leq i < n-1; \mathtt{B}(n-1) \to \mathtt{C}() \}$

$\Rightarrow$ fixes order of reduction

Allow reordering of updates:

- read in C should depend on all updates in B
  - $\Rightarrow$ updates should not kill dependences
  - $\Rightarrow$ remove updates from must-writes
- updates should not depend on each other
  - $\Rightarrow$ remove false dependences between updates that flow to the same read, provided the read does not also depend on intermediate writes

Dataflow: $\{ \mathtt{A}() \to \mathtt{B}(i) : 0 \leq i < n; \mathtt{B}(i) \to \mathtt{C}() : 0 \leq i < n \}$

Outline

Aliasing

Some possible ways of handling aliasing:

- use an input language that does not permit aliasing
- pretend the problem does not exist
- require user to ensure absence of aliasing
  $\Rightarrow$ e.g., use `restrict` keyword
- handle as may-write
  $\Rightarrow$ may lead to too many dependences
- check aliasing at run-time
  $\Rightarrow$ use original code in case of aliasing

Outline

Cardinality

- Cardinality of a set
  - $\Rightarrow$ number of elements in the set
  - $\Rightarrow$ may depend on symbolic constants

$S = \{ S(\mathbf{i}) : f(\mathbf{i}) \}$

$\operatorname{card} S = \{ n : n = \#\mathbf{i} : f(\mathbf{i}) \}$ $\qquad \operatorname{card}\left(\bigcup_i S_i\right) := \sum_i \operatorname{card} S_i$

$\operatorname{card} \{ \mathtt{A}(i) : 0 \leq i \leq n; \mathtt{B}() \} = n + 2$

- Cardinality of a binary relation
  - $\Rightarrow$ for each domain element, number of corresponding images

$R = \{ S(\mathbf{i}) \to T(\mathbf{j}) : f(\mathbf{i}, \mathbf{j}) \}$

$\operatorname{card} R = \{ S(\mathbf{i}) \to n : n = \#\mathbf{j} : f(\mathbf{i}, \mathbf{j}) \}$ $\quad \operatorname{card}\left(\bigcup_i R_i\right) := \sum_i \operatorname{card} R_i$

$R = \{ \mathtt{A}(i) \to \mathtt{C}(i) : 0 \leq i \leq n; \mathtt{B}() \to \mathtt{C}(i) : 0 \leq i \leq n \}$
$\operatorname{card} R = \{ \mathtt{A}(i) \to 1 : 0 \leq i \leq n; \mathtt{B}() \to n + 1 \}$

$\Rightarrow$ not a Presburger formula

Cardinality Examples

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

- How many times is the statement executed?

  $\mathrm{card}\{(i,j) : 0 \leq i < N \wedge 0 \leq j < N - i\}$

  $\Rightarrow \{\frac{N+N^2}{2} : N \geq 1\}$

Cardinality Examples

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

- How many times is the statement executed?

  $\mathrm{card}\{\,(i,j) : 0 \le i < N \wedge 0 \le j < N - i\,\}$

  $\Rightarrow \{\,\frac{N+N^2}{2} : N \ge 1\,\}$
- How many times is a given array element written?

  $\mathrm{card}\left(\{\,(i,j) \to \mathsf{a}(i+j) : 0 \le i < N \wedge 0 \le j < N - i\,\}\right)^{-1}$

  $\Rightarrow \{\,\mathsf{a}(a) \to 1 + a : 0 \le a < N\,\}$

Cardinality Examples

```
for (i = 0; i < N; ++i)
    for (j = 0; j < N - i; ++j)
        a[i+j] = f(a[i+j]);
```

- How many times is the statement executed?

  $\mathrm{card}\{(i,j) : 0 \le i < N \wedge 0 \le j < N - i\}$

  $\Rightarrow \{\frac{N+N^2}{2} : N \ge 1\}$

- How many times is a given array element written?

  $\mathrm{card}\left(\{(i,j) \to \mathsf{a}(i+j) : 0 \le i < N \wedge 0 \le j < N - i\}\right)^{-1}$

  $\Rightarrow \{\mathsf{a}(a) \to 1 + a : 0 \le a < N\}$

- How many array elements are written?

  $\mathrm{card}\left(\mathrm{ran}\{(i,j) \to \mathsf{a}(i+j) : 0 \le i < N \wedge 0 \le j < N - i\}\right)$

  $\Rightarrow \{N : N \ge 1\}$

Cardinality Examples (2)

How many times is S1 executed ?

```
for (i = max(0,N-M); i <= N-M+3; i++)
    for (j = 0; j <= N-2*i; j++)
        S1;
```

$$\mathrm{card}\{(i,j) : 0, N-M \leq i \leq N-M+3 \wedge 0 \leq j \leq N-2i\}$$

$$\begin{cases} -4N+8M-8 & \text{if } M \leq N \leq 2M-6 \\ MN-2N-M^2+6M-8 & \text{if } N \leq M \leq N+3 \wedge N \leq 2M \\ \frac{N^2}{4}+\frac{3}{4}N+\frac{1}{2}\left\lfloor\frac{N}{2}\right\rfloor+1 & \text{if } 0 \leq N \leq M \wedge 2M \leq N+6 \\ \frac{N^2}{4}-MN-\frac{5}{4}N+M^2+2M+\frac{1}{2}\left\lfloor\frac{N}{2}\right\rfloor+1 & \text{if } M \leq N \leq 2M \leq N+6 \end{cases}$$

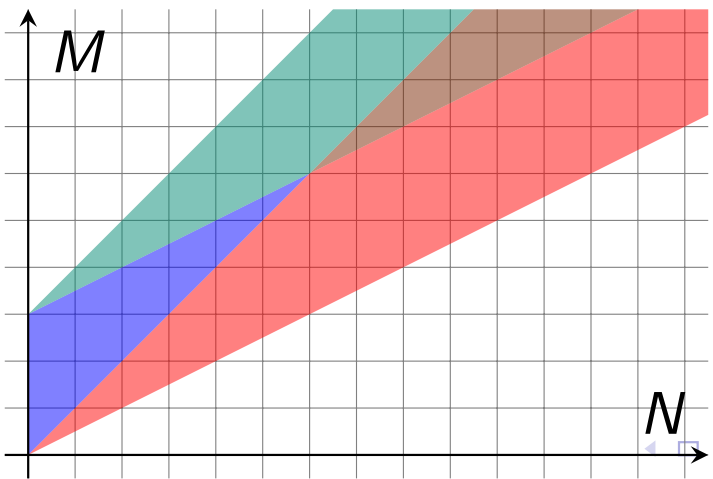
Cardinality Examples (2)

How many times is S1 executed ?

```
for (i = max(0,N-M); i <= N-M+3; i++)
    for (j = 0; j <= N-2*i; j++)
        S1;
```

$$\mathrm{card}\{(i,j) : 0, N-M \leq i \leq N-M+3 \wedge 0 \leq j \leq N-2i\}$$

$$\begin{cases} -4N+8M-8 & \text{if } M \leq N \leq 2M-6 \\ MN-2N-M^2+6M-8 & \text{if } N \leq M \leq N+3 \wedge N \leq 2M \\ \frac{N^2}{4}+\frac{3}{4}N+\frac{1}{2}\left\lfloor\frac{N}{2}\right\rfloor+1 & \text{if } 0 \leq N \leq M \wedge 2M \leq N+6 \\ \frac{N^2}{4}-MN-\frac{5}{4}N+M^2+2M+\frac{1}{2}\left\lfloor\frac{N}{2}\right\rfloor+1 & \text{if } M \leq N \leq 2M \leq N+6 \end{cases}$$



Cardinality Representation

- Integer quasi affine expression $\lfloor x/2 \rfloor + 3N$

  $\Rightarrow$ Presburger term

  That is, a term constructed from variables, symbolic constants, integer constants, addition $(+)$, subtraction $(-)$ and integer division by a constant $(\lfloor \cdot/d \rfloor)$

Cardinality Representation

- Integer quasi affine expression $\lfloor x/2 \rfloor + 3N$
  - $\Rightarrow$ Presburger term
    That is, a term constructed from variables, symbolic constants, integer constants, addition $(+)$, subtraction $(-)$ and integer division by a constant $(\lfloor \cdot/d \rfloor)$
- Rational polynomial expression $x^2 - N/2$
  - $\Rightarrow$ a term constructed from variables, symbolic constants, rational constants, addition $(+)$, subtraction $(-)$ and multiplication $(\cdot)$

Cardinality Representation

- Integer quasi affine expression $\lfloor x/2 \rfloor + 3N$
  - $\Rightarrow$ Presburger term
    That is, a term constructed from variables, symbolic constants, integer constants, addition $(+)$, subtraction $(-)$ and integer division by a constant $(\lfloor \cdot/d \rfloor)$
- Rational polynomial expression $x^2 - N/2$
  - $\Rightarrow$ a term constructed from variables, symbolic constants, rational constants, addition $(+)$, subtraction $(-)$ and multiplication $(\cdot)$
- Quasi polynomial expression $(\lfloor x/2 \rfloor + 3N)^2 - N/2$
  - $\Rightarrow$ a rational polynomial expression with variables replaced by integer quasi affine expressions

Cardinality Representation

- Integer quasi affine expression $\lfloor x/2 \rfloor + 3N$
  - $\Rightarrow$ Presburger term
    That is, a term constructed from variables, symbolic constants, integer constants, addition $(+)$, subtraction $(-)$ and integer division by a constant $(\lfloor \cdot/d \rfloor)$
- Rational polynomial expression $x^2 - N/2$
  - $\Rightarrow$ a term constructed from variables, symbolic constants, rational constants, addition $(+)$, subtraction $(-)$ and multiplication $(\cdot)$
- Quasi polynomial expression $(\lfloor x/2 \rfloor + 3N)^2 - N/2$
  - $\Rightarrow$ a rational polynomial expression with variables replaced by integer quasi affine expressions
- Piecewise quasi affine/polynomial expression
  - $\Rightarrow$ a list of pairs of pairs of Presburger sets and quasi affine/polynomial expressions $E = (S_i, e_i)_i$, with S_i disjoint

$$E(\mathbf{j}) = \begin{cases} e_i(\mathbf{j}) & \text{if } \mathbf{j} \in S_i \\ \perp/0 & \text{otherwise} \end{cases}$$

Cardinality Representation

- Integer quasi affine expression $\lfloor x/2 \rfloor + 3N$
  - $\Rightarrow$ Presburger term
    That is, a term constructed from variables, symbolic constants, integer constants, addition $(+)$, subtraction $(-)$ and integer division by a constant $(\lfloor \cdot/d \rfloor)$
- Rational polynomial expression $x^2 - N/2$
  - $\Rightarrow$ a term constructed from variables, symbolic constants, rational constants, addition $(+)$, subtraction $(-)$ and multiplication $(\cdot)$
- Quasi polynomial expression $(\lfloor x/2 \rfloor + 3N)^2 - N/2$
  - $\Rightarrow$ a rational polynomial expression with variables replaced by integer quasi affine expressions
- Piecewise quasi affine/polynomial expression
  - $\Rightarrow$ a list of pairs of pairs of Presburger sets and quasi affine/polynomial expressions $E = (S_i, e_i)_i$, with S_i disjoint

$$E(\mathbf{j}) = \begin{cases} e_i(\mathbf{j}) & \text{if } \mathbf{j} \in S_i \\ \perp/0 & \text{otherwise} \end{cases}$$

Note: in practice, cardinality result does not contain nested integer divisions

Basic Operations on Piecewise Expressions

- Piecewise (rational) quasi affine expressions
  - addition $(+)$
  - subtraction $(-)$
  - negation $(-)$
  - minimum (min), maximum (max)
  - multiplication by constant $(\cdot d)$
  - division by constant $(/d)$
  - remainder on integer division by constant $(\bmod d)$
  - floor $(\lfloor \cdot \rfloor)$
  - ceiling $(\lceil \cdot \rceil)$

Basic Operations on Piecewise Expressions

- Piecewise (rational) quasi affine expressions
  - addition $(+)$
  - subtraction $(-)$
  - negation $(-)$
  - minimum (min), maximum (max)
  - multiplication by constant $(\cdot d)$
  - division by constant $(/d)$
  - remainder on integer division by constant $(\bmod d)$
  - floor $(\lfloor \cdot \rfloor)$
  - ceiling $(\lceil \cdot \rceil)$
- Piecewise quasi polynomial expressions
  - addition $(+)$
  - subtraction $(-)$
  - negation $(-)$
  - multiplication $(\cdot)$
  - exponentiation by positive integer constant $(\cdot^d)$

Bounds on Piecewise Quasi Polynomials

$$m(N) = \max_{(x,y):x,y\geq 0 \wedge x+y\leq N} 4+x+y-(x-2)^2$$

Question

Can exact maximum be computed in general?

Bounds on Piecewise Quasi Polynomials

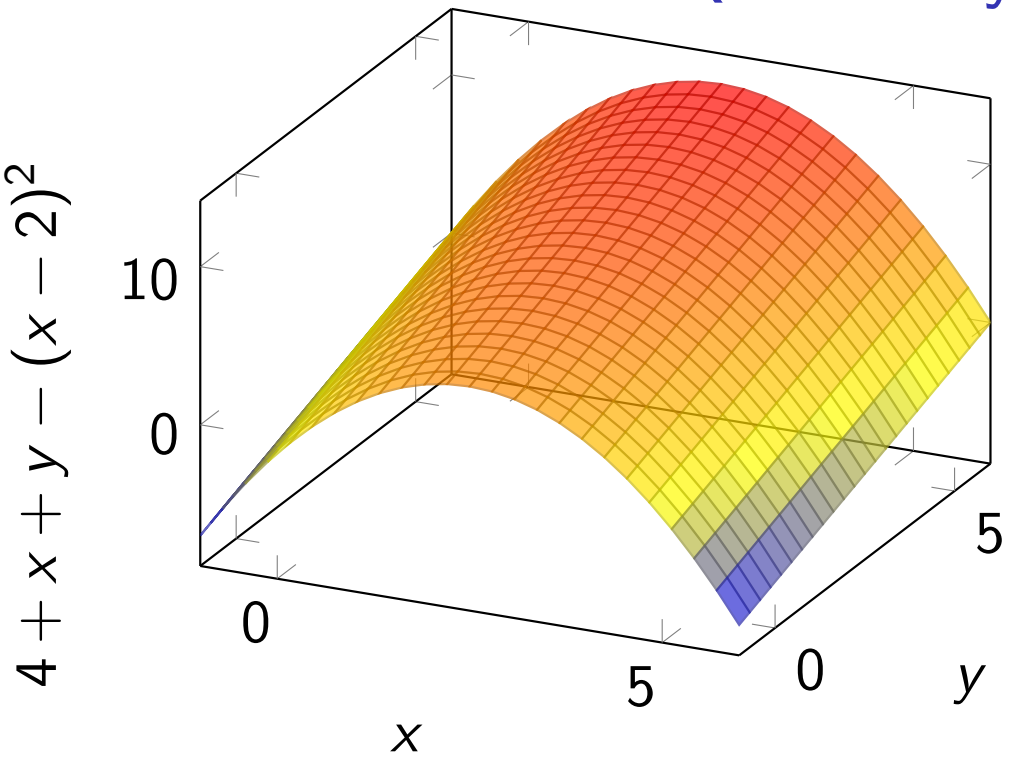
$$m(N) = \max_{(x,y):x,y\geq 0 \wedge x+y\leq N} 4+x+y-(x-2)^2$$

Question

Can exact maximum be computed in general?

Upper bound $u(N) \geq m(N)$ can be computed

Bounds on Piecewise Quasi Polynomials



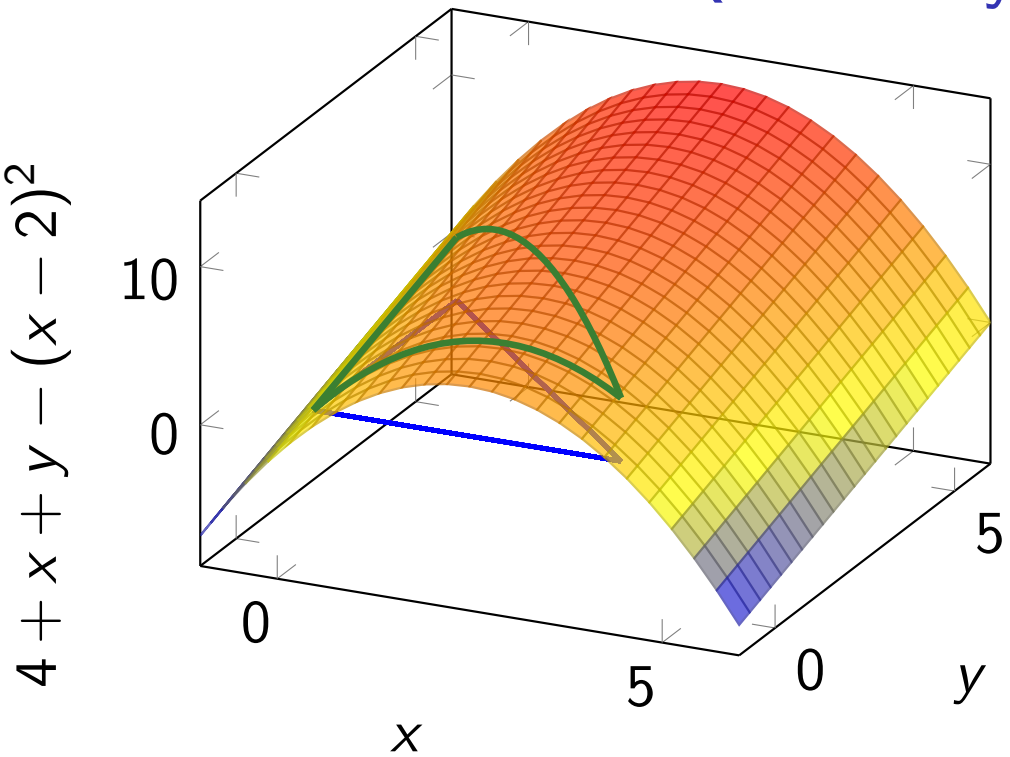
$$m(N) = \max_{(x,y):x,y\geq 0 \wedge x+y\leq N} 4+x+y-(x-2)^2$$

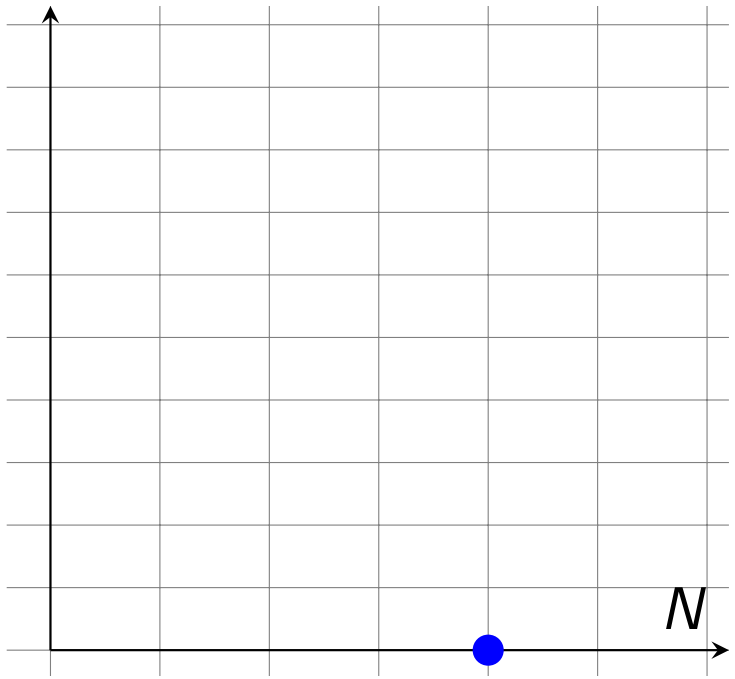
Question

Can exact maximum be computed in general?

Upper bound $u(N) \geq m(N)$ can be computed

Bounds on Piecewise Quasi Polynomials





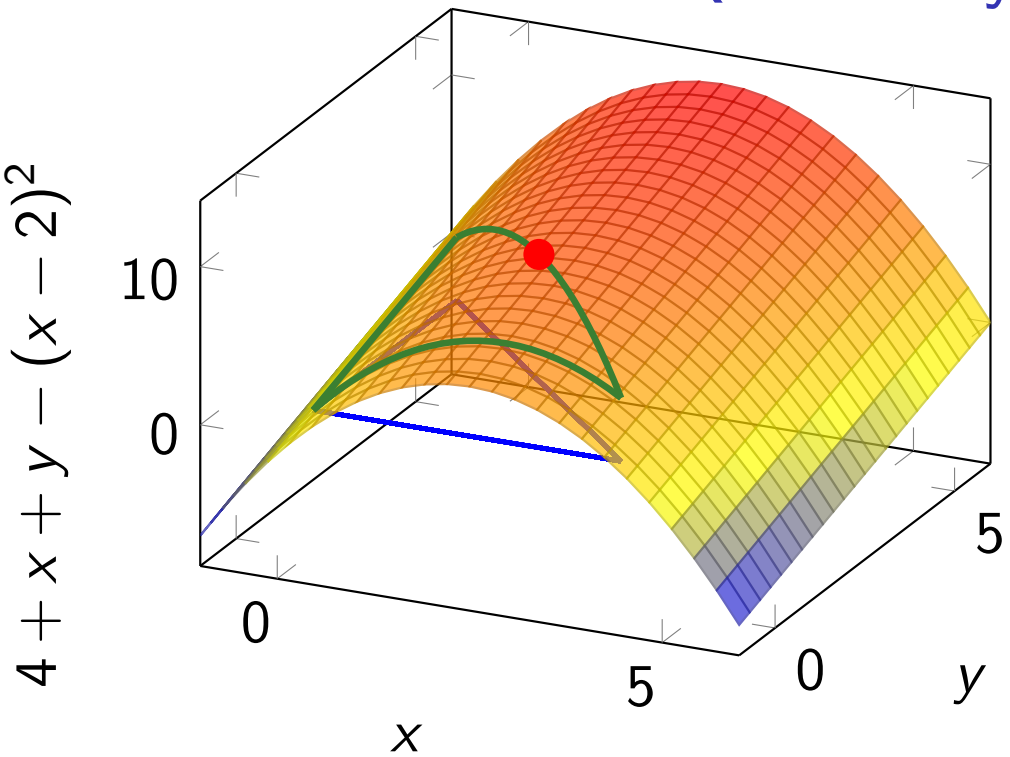
$$m(N) = \max_{(x,y):x,y\geq 0 \wedge x+y\leq N} 4+x+y-(x-2)^2$$

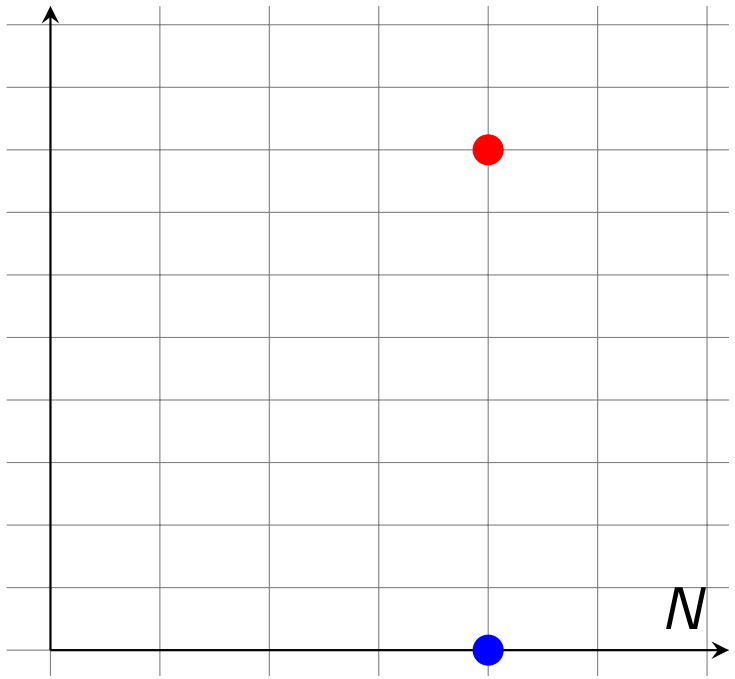
Question

Can exact maximum be computed in general?

Upper bound $u(N) \geq m(N)$ can be computed

Bounds on Piecewise Quasi Polynomials





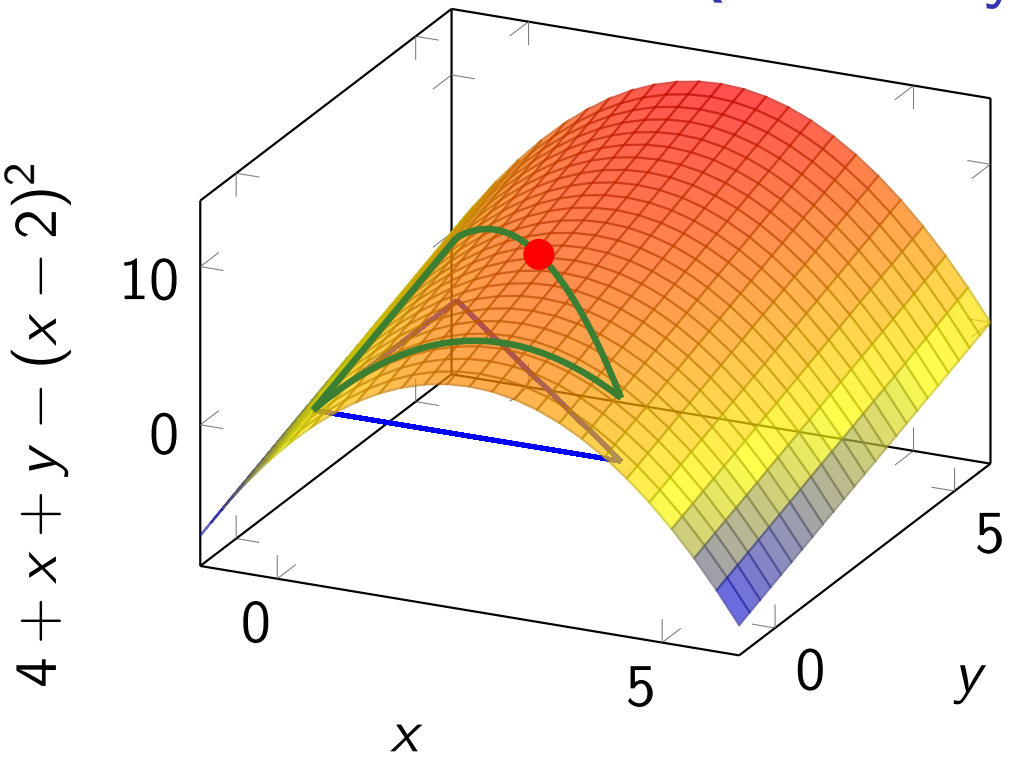
$$m(N) = \max_{(x,y):x,y\geq 0 \wedge x+y\leq N} 4+x+y-(x-2)^2$$

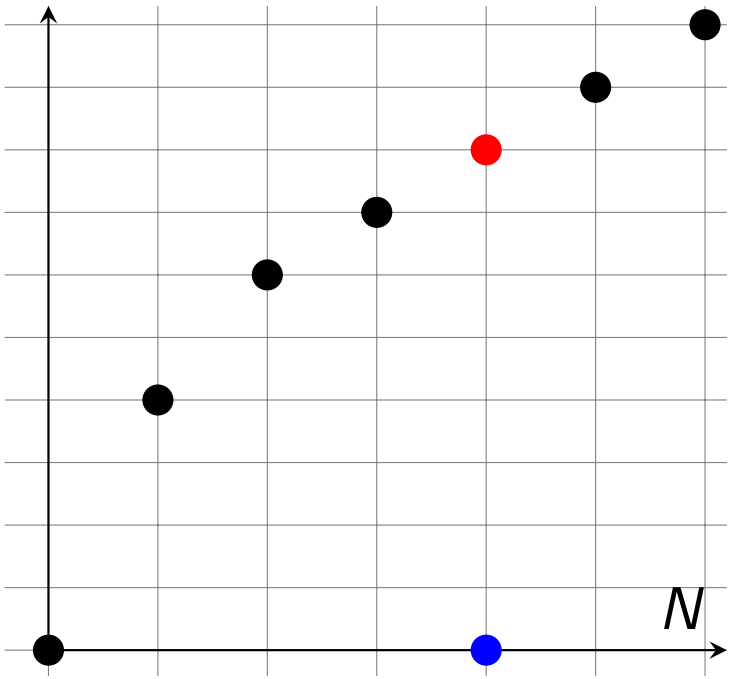
Question

Can exact maximum be computed in general?

Upper bound $u(N) \geq m(N)$ can be computed

Bounds on Piecewise Quasi Polynomials





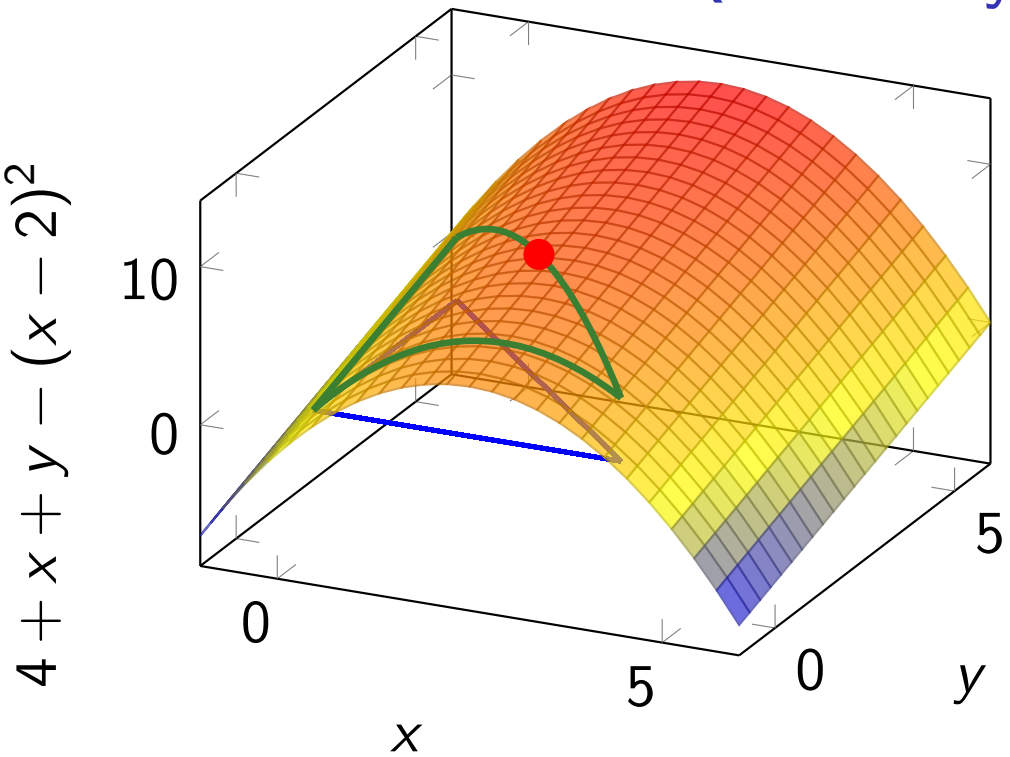
$$m(N) = \max_{(x,y):x,y\geq 0 \wedge x+y\leq N} 4+x+y-(x-2)^2$$

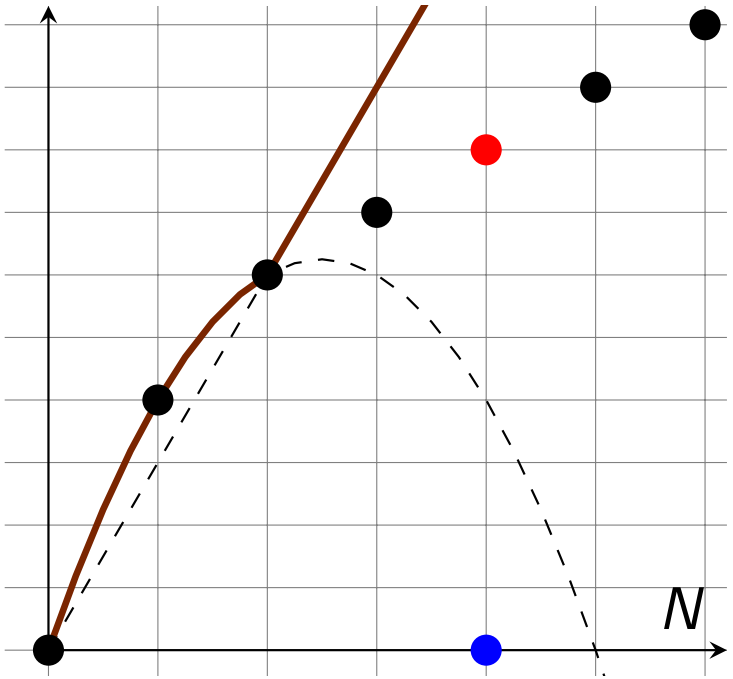
Question

Can exact maximum be computed in general?

Upper bound $u(N) \geq m(N)$ can be computed

Bounds on Piecewise Quasi Polynomials





$$m(N) = \max_{(x,y):x,y\geq 0 \wedge x+y\leq N} 4+x+y-(x-2)^2 \leq u(N) = \max(3N, 5N - N^2)$$

Question

Can exact maximum be computed in general?

Upper bound $u(N) \geq m(N)$ can be computed

Bounds on Piecewise Quasi Polynomials — Example

```
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j) {
        p = malloc(i * j + i - N + 1);
        /* ... */
        free(p);
    }
```

How much memory is needed?

Bounds on Piecewise Quasi Polynomials — Example

```
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j) {
        p = malloc(i * j + i - N + 1);
        /* ... */
        free(p);
    }
```

How much memory is needed?

$$\mathrm{ub}\begin{cases} ij + i - N + 1 & \text{if } 0 \leq i < N \wedge i \leq j < N \end{cases}$$

Bounds on Piecewise Quasi Polynomials — Example

```
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j) {
        p = malloc(i * j + i - N + 1);
        /* ... */
        free(p);
    }
```

How much memory is needed?

$$\mathrm{ub}\begin{cases} ij + i - N + 1 & \text{if } 0 \le i < N \wedge i \le j < N \end{cases}$$

Result:

$$\begin{cases} \max(1 - 2N + N^2) & \text{if } N \ge 1 \end{cases}$$

(exact maximum)

Maximal Number of Live Memory elements

- Assume each statement instance writes to at most one array element
  $\Rightarrow$ Each live element can be identified by write instance
- Compute dataflow relation D
- For each write instance compute last read
  $L = O^{-1} \circ \operatorname{lexmax}(O \circ D)$
- For each statement instance **i**, count write instances that precede **i** such that corresponding last read follows **i**
  $\Rightarrow$ Number of live elements at **i**
  $N = \operatorname{card}\left(((O \succ O) \cap_{ran} (\operatorname{dom} L)) \cap \left(L^{-1} \circ (O \preccurlyeq O)\right)\right)$
- Compute upper bound
  $U = \operatorname{ub} N$

Maximal Number of Live Memory elements — Example

```
for (i = 0; i < N; ++i)
S1:     t[i] = f(a[i]);
for (i = 0; i < N; ++i)
S2:     b[i] = g(t[N-i-1]);
```

$$I = \{ \mathrm{S1}(i) : 0 \leq i < N; \mathrm{S2}(i) : 0 \leq i < N \}$$

$$O = \{ \mathrm{S1}(i) \rightarrow (0, i); \mathrm{S2}(i) \rightarrow (1, i) \}$$

$$D = \{ \mathrm{S1}(i) \rightarrow \mathrm{S2}(N - 1 - i) : 0 \leq i < N \}$$

Maximal Number of Live Memory elements — Example

```
for (i = 0; i < N; ++i)
S1:     t[i] = f(a[i]);
for (i = 0; i < N; ++i)
S2:     b[i] = g(t[N-i-1]);
```

$$I = \{\, \mathrm{S1}(i) : 0 \le i < N; \mathrm{S2}(i) : 0 \le i < N \,\}$$

$$O = \{\, \mathrm{S1}(i) \to (0, i); \mathrm{S2}(i) \to (1, i) \,\}$$

$$D = \{\, \mathrm{S1}(i) \to \mathrm{S2}(N - 1 - i) : 0 \le i < N \,\}$$

$$\begin{aligned} L &= O^{-1} \circ \mathrm{lexmax}(O \circ D) \\ &= \{\, \mathrm{S1}(i) \to \mathrm{S2}(N - 1 - i) : 0 \le i < N \,\} \end{aligned}$$

$$\begin{aligned} N &= \mathrm{card}\left(((O \succ O) \cap_{\mathsf{ran}} (\mathrm{dom}\, L)) \cap \left(L^{-1} \circ (O \preccurlyeq O)\right)\right) \\ &= \{\, \mathrm{S1}(i) \to i : 1 \le i < N; \mathrm{S2}(i) \to N - i : 0 \le i < N \,\} \end{aligned}$$

$$\begin{aligned} U &= \mathrm{ub}\, N \\ &= \{\, \mathsf{max}(N) : N \ge 1 \,\} \end{aligned}$$

Weighted Counting

$$G = F \circ R$$

with F a piecewise quasi polynomial and R a Presburger relation is a piecewise quasi polynomial G such that

$$G(\mathbf{i}) = \sum_{\mathbf{j}:R(\mathbf{i},\mathbf{j})} F(\mathbf{j})$$

Weighted Counting

$$G = F \circ R = \left\{ (x, y) \to \frac{x^2 + y^2}{4} : 1 \leq x, y \leq 2 \right\} \circ \{ (x) \to (x, y) \}$$

with F a piecewise quasi polynomial and R a Presburger relation is a piecewise quasi polynomial G such that

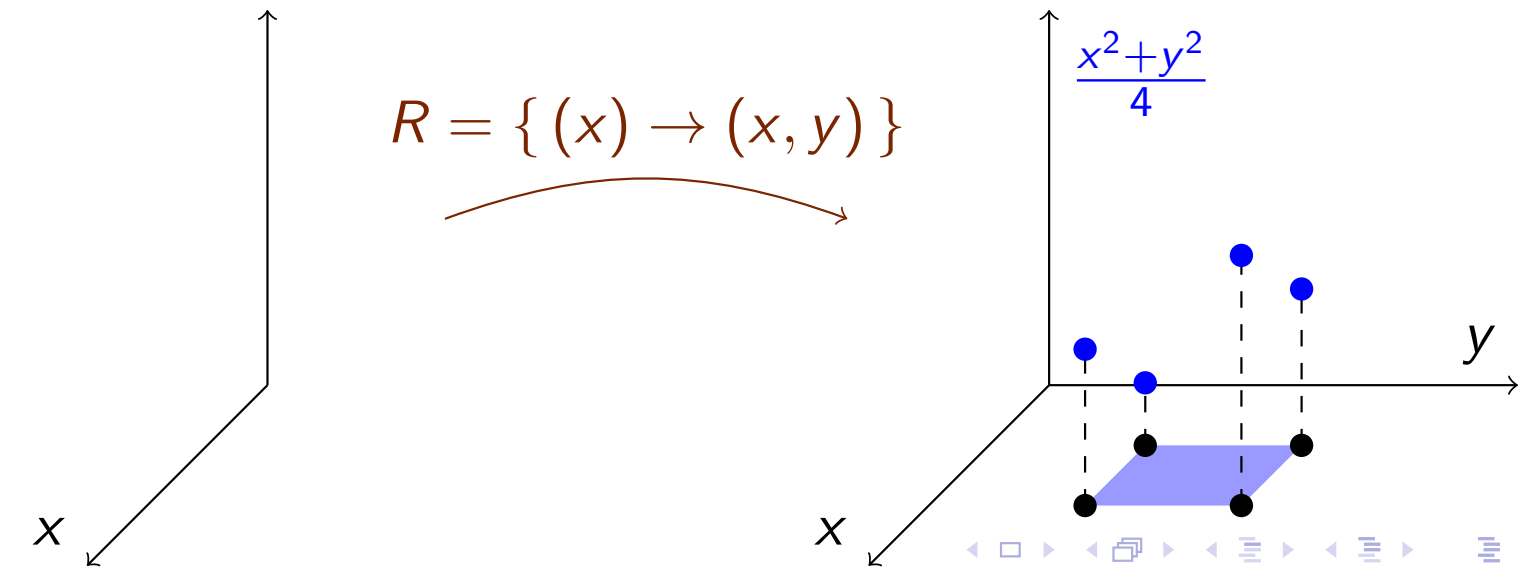
$$G(\mathbf{i}) = \sum_{\mathbf{j}:R(\mathbf{i},\mathbf{j})} F(\mathbf{j})$$

Weighted Counting

$$G = F \circ R = \left\{ (x, y) \to \frac{x^2 + y^2}{4} : 1 \leq x, y \leq 2 \right\} \circ \{ (x) \to (x, y) \}$$

with F a piecewise quasi polynomial and R a Presburger relation is a piecewise quasi polynomial G such that

$$G(\mathbf{i}) = \sum_{\mathbf{j}:R(\mathbf{i},\mathbf{j})} F(\mathbf{j})$$



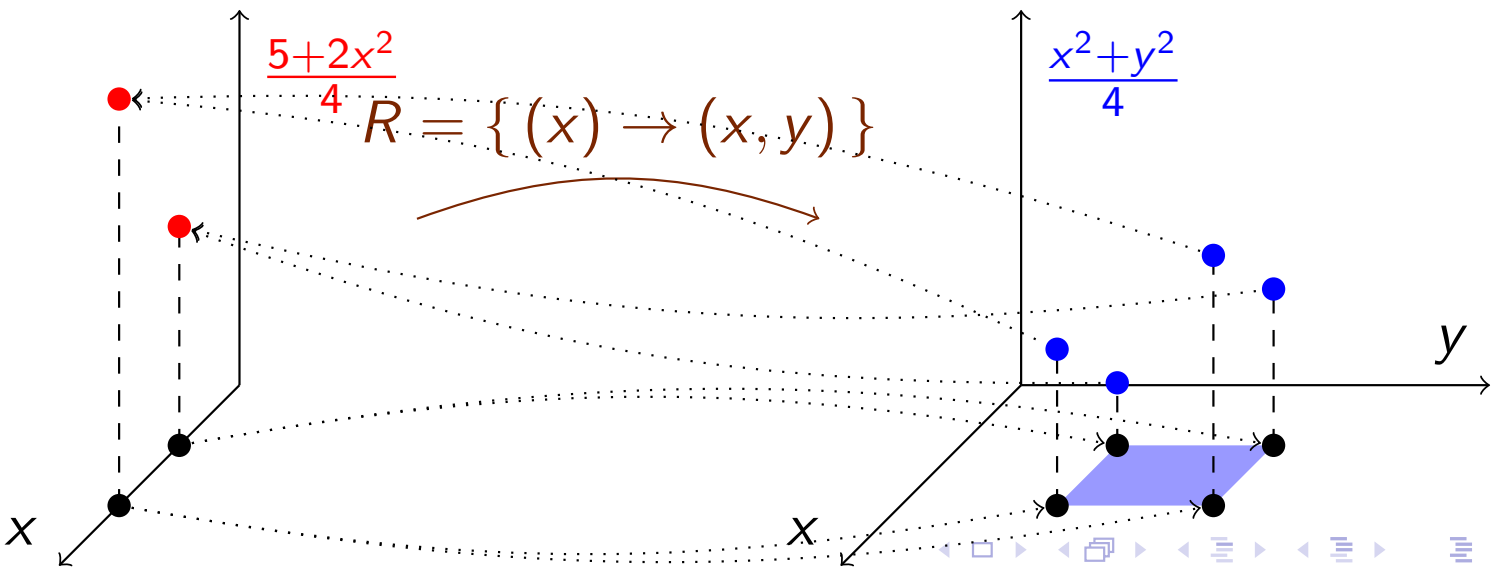
Weighted Counting

$$G = F \circ R = \left\{ (x,y) \to \frac{x^2+y^2}{4} : 1 \le x,y \le 2 \right\} \circ \{ (x) \to (x,y) \}$$

$$= \left\{ (x) \to \frac{5+2x^2}{2} : 1 \le x \le 2 \right\}$$

with F a piecewise quasi polynomial and R a Presburger relation is a piecewise quasi polynomial G such that

$$G(\mathbf{i}) = \sum_{\mathbf{j}:R(\mathbf{i},\mathbf{j})} F(\mathbf{j})$$



Compositions with Piecewise (Folds of) Quasi polynomials

$$F \circ R$$

- R: $D_1 \to D_2$ is a Presburger relation
- F: $D_2 \to \mathbb{Q}$ may be
  - piecewise quasi polynomial
    (result of counting problem)

    $\Rightarrow$ take sum over $(\operatorname{ran} R) \cap (\operatorname{dom} F)$
  - piecewise fold of quasi polynomials
    (result of upper bound computation)

    $\Rightarrow$ compute bound over $(\operatorname{ran} R) \cap (\operatorname{dom} F)$
- $(F \circ R)$: $D_1 \to \mathbb{Q}$ of same type as F

if R is single-valued, then sum/bound is computed over a single point

Compositions with Piecewise (Folds of) Quasi polynomials

$$F \circ R \qquad \text{or} \qquad F(S)$$

- R: $D_1 \to D_2$ is a Presburger relation
- $S \subseteq D_2$ is a Presburger set
- F: $D_2 \to \mathbb{Q}$ may be
  - piecewise quasi polynomial
    (result of counting problem)

    $\Rightarrow$ take sum over $(\operatorname{ran} R) \cap (\operatorname{dom} F)$ or $S \cap (\operatorname{dom} F)$
  - piecewise fold of quasi polynomials
    (result of upper bound computation)

    $\Rightarrow$ compute bound over $(\operatorname{ran} R) \cap (\operatorname{dom} F)$ or $S \cap (\operatorname{dom} F)$
- $(F \circ R)$: $D_1 \to \mathbb{Q}$ of same type as F
- $F(S)$: $\mathbb{Q}$ of same type as F

if R is single-valued, then sum/bound is computed over a single point

Example: Total Memory Allocation

```
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j)
        p[i][j] = malloc(i * j + i - N + 1);
/* ... */
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j)
        free(p[i][j]);
```

How much memory allocated in total?

Example: Total Memory Allocation

```
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j)
        p[i][j] = malloc(i * j + i - N + 1);
/* ... */
for (i = 0; i < N; ++i)
    for (j = i; j < N; ++j)
        free(p[i][j]);
```

How much memory allocated in total?

$$F = \{ (i,j) \to ij + i - N + 1 \}$$

$$I = \{ (i,j) : 0 \leq i < N \wedge i \leq j < N \}$$

$$F(I) = \left\{ \frac{5}{12}N - \frac{1}{8}N^2 - \frac{5}{12}N^3 + \frac{1}{8}N^4 : N \geq 1 \right\}$$

[7]

Dynamic Memory Requirement Estimation

How much memory is needed to execute the following program?

```
void m0(int m) {
  for (c = 0; c < m; c++) {
    m1(c);                    /*S1*/
    B[] m2Arr = m2(2*m-c);/*S2*/
  }
}
void m1(int k) {
  for (i = 1; i <= k; i++) {
    A a = new A();            /*S3*/
    B[] dummyArr = m2(i); /*S4*/
  }
}
B[] m2(int n) {
  B[] arrB = new B[n];      /*S5*/
  for (j = 1; j <= n; j++)
    B b = new B();            /*S6*/
  return arrB;
}
```

[7]

Dynamic Memory Requirement Estimation

How much memory is needed to execute the following program?

```
void m0(int m) {
  for (c = 0; c < m; c++) {
    m1(c);                          /*S1*/
    B[] m2Arr = m2(2*m-c);/*S2*/
  }
}
void m1(int k) {
  for (i = 1; i <= k; i++) {
    A a = new A();                  /*S3*/
    B[] dummyArr = m2(i); /*S4*/
  }
}
B[] m2(int n) {
  B[] arrB = new B[n];              /*S5*/
  for (j = 1; j <= n; j++)
    B b = new B();                  /*S6*/
  return arrB;
}
```

$$I = \{ \mathtt{m0}(m) \rightarrow \mathtt{S1}(c) : 0 \leq c < m;$$
$$\mathtt{m0}(m) \rightarrow \mathtt{S2}(c) : 0 \leq c < m;$$
$$\mathtt{m1}(k) \rightarrow \mathtt{S3}(i) : 1 \leq i \leq k;$$
$$\mathtt{m1}(k) \rightarrow \mathtt{S4}(i) : 1 \leq i \leq k;$$
$$\mathtt{m2}(n) \rightarrow \mathtt{S5}();$$
$$\mathtt{m2}(n) \rightarrow \mathtt{S6}(j) : 1 \leq j \leq n \}$$

[7]

Dynamic Memory Requirement Estimation

How much (scoped) memory is needed?

$\Rightarrow$ compute for each method

$\mathrm{ret}_{\mathtt{m}}$ size of memory returned by `m`

$\mathrm{cap}_{\mathtt{m}}$ size of memory "captured" (not returned) by `m`

$\mathrm{memRq}_{\mathtt{m}}$ total memory requirements of `m`

$$\mathrm{ret}_{\mathtt{m}} + \mathrm{cap}_{\mathtt{m}} = \sum_{\mathtt{p} \text{ called by } \mathtt{m}} \mathrm{ret}_{\mathtt{p}}$$

$$\mathrm{memRq}_{\mathtt{m}} = \mathrm{cap}_{\mathtt{m}} + \max_{\mathtt{p} \text{ called by } \mathtt{m}} \mathrm{memRq}_{\mathtt{p}}$$

[7]

Dynamic Memory Requirement Estimation

How much (scoped) memory is needed?

$\Rightarrow$ compute for each method

$\mathrm{ret}_{\mathtt{m}}$ size of memory returned by $\mathtt{m}$

$\mathrm{cap}_{\mathtt{m}}$ size of memory "captured" (not returned) by $\mathtt{m}$

$\mathrm{memRq}_{\mathtt{m}}$ total memory requirements of $\mathtt{m}$

$$\mathrm{ret}_{\mathtt{m}} + \mathrm{cap}_{\mathtt{m}} = \sum_{\mathtt{p} \text{ called by } \mathtt{m}} \mathrm{ret}_{\mathtt{p}}$$

$$\mathrm{memRq}_{\mathtt{m}} = \mathrm{cap}_{\mathtt{m}} + \max_{\mathtt{p} \text{ called by } \mathtt{m}} \mathrm{memRq}_{\mathtt{p}}$$

$\Rightarrow$ summarize over iteration domain, i.e., compose with $M = (\underrightarrow{\mathrm{dom}}\, I)^{-1}$

$$\begin{aligned} M = \{ & \mathtt{m0}(m) \to (\mathtt{m0}(m) \to \mathtt{S1}(c)) : 0 \le c < m; \\ & \mathtt{m0}(m) \to (\mathtt{m0}(m) \to \mathtt{S2}(c)) : 0 \le c < m; \\ & \mathtt{m1}(k) \to (\mathtt{m1}(k) \to \mathtt{S3}(i)) : 1 \le i \le k; \\ & \mathtt{m1}(k) \to (\mathtt{m1}(k) \to \mathtt{S4}(i)) : 1 \le i \le k; \\ & \mathtt{m2}(n) \to (\mathtt{m2}(n) \to \mathtt{S5}()); \mathtt{m2}(n) \to (\mathtt{m2}(n) \to \mathtt{S6}(j)) : 1 \le j \le n \} \end{aligned}$$

[7]

Dynamic Memory Requirement Estimation

$$\mathrm{ret}_{\mathtt{m}} + \mathrm{cap}_{\mathtt{m}} = \sum_{\mathtt{p} \text{ called by } \mathtt{m}} \mathrm{ret}_{\mathtt{p}}$$

$$\mathrm{memRq}_{\mathtt{m}} = \mathrm{cap}_{\mathtt{m}} + \max_{\mathtt{p} \text{ called by } \mathtt{m}} \mathrm{memRq}_{\mathtt{p}}$$

[7]

Dynamic Memory Requirement Estimation

$$\mathrm{ret_m} + \mathrm{cap_m} = \sum_{\mathtt{p} \text{ called by } \mathtt{m}} \mathrm{ret_p}$$

$$\mathrm{memRq_m} = \mathrm{cap_m} + \max_{\mathtt{p} \text{ called by } \mathtt{m}} \mathrm{memRq_p}$$

```
B[] m2(int n) {
 B[] arrB = new B[n];        /*S5*/
 for (j = 1; j <= n; j++)
  B b = new B();             /*S6*/
 return arrB;
}
```

[7]

Dynamic Memory Requirement Estimation

$$\mathrm{ret}_{\mathtt{m}} + \mathrm{cap}_{\mathtt{m}} = \sum_{\mathtt{p}\ \text{called by}\ \mathtt{m}} \mathrm{ret}_{\mathtt{p}}$$

$$\mathrm{memRq}_{\mathtt{m}} = \mathrm{cap}_{\mathtt{m}} + \max_{\mathtt{p}\ \text{called by}\ \mathtt{m}} \mathrm{memRq}_{\mathtt{p}}$$

```
B[] m2(int n) {
 B[] arrB = new B[n];          /*S5*/
 for (j = 1; j <= n; j++)
  B b = new B();               /*S6*/
 return arrB;
}
```

$$\mathrm{ret}_{\mathtt{m2}} = \{\, (\mathtt{m2}(n) \to \mathtt{S5}()) \to n : n \geq 0 \,\} \circ M$$

$$\mathrm{cap}_{\mathtt{m2}} = \{\, (\mathtt{m2}(n) \to \mathtt{S6}(j)) \to 1 \,\} \circ M$$

$$\mathrm{memRq}_{\mathtt{m2}} = \mathrm{cap}_{\mathtt{m2}} + \{\, \mathtt{m2}(n) \to \max(0) \,\}$$

[7]

Dynamic Memory Requirement Estimation

$$\mathrm{ret}_{\mathtt{m}} + \mathrm{cap}_{\mathtt{m}} = \sum_{\mathtt{p}\ \text{called by}\ \mathtt{m}} \mathrm{ret}_{\mathtt{p}}$$

$$\mathrm{memRq}_{\mathtt{m}} = \mathrm{cap}_{\mathtt{m}} + \max_{\mathtt{p}\ \text{called by}\ \mathtt{m}} \mathrm{memRq}_{\mathtt{p}}$$

```
B[] m2(int n) {
 B[] arrB = new B[n];        /*S5*/
 for (j = 1; j <= n; j++)
  B b = new B();             /*S6*/
 return arrB;
}
```

$$\mathrm{ret}_{\mathtt{m2}} = \{\, (\mathtt{m2}(n) \to \mathtt{S5}()) \to n : n \geq 0 \,\} \circ M = \{\, \mathtt{m2}(n) \to n : n \geq 0 \,\}$$

$$\mathrm{cap}_{\mathtt{m2}} = \{\, (\mathtt{m2}(n) \to \mathtt{S6}(j)) \to 1 \,\} \circ M = \{\, \mathtt{m2}(n) \to n : n \geq 1 \,\}$$

$$\mathrm{memRq}_{\mathtt{m2}} = \mathrm{cap}_{\mathtt{m2}} + \{\, \mathtt{m2}(n) \to \max(0) \,\} = \{\, \mathtt{m2}(n) \to \max(n) : n \geq 1 \,\}$$

[7]

Dynamic Memory Requirement Estimation

```
void m1(int k) {
    for (i = 1; i <= k; i++) {
        A a = new A();                    /* S3 */
        B[] dummyArr = m2(i);             /* S4 */
    }
}
```

$$\mathrm{cap}_{\mathtt{m1}}(k) = \sum_{1 \le i \le k} (1 + \mathrm{ret}_{\mathtt{m2}}(i))$$

$\mathrm{ret}_{\mathtt{m2}}$ is a function of the arguments of `m2`

We want to use it as a function of the arguments and local variables of `m1`

[7]

Dynamic Memory Requirement Estimation

```
void m1(int k) {
    for (i = 1; i <= k; i++) {
        A a = new A();                    /* S3 */
        B[] dummyArr = m2(i);             /* S4 */
    }
}
```

$$\mathrm{cap}_{\mathtt{m1}}(k) = \sum_{1 \le i \le k} (1 + \mathrm{ret}_{\mathtt{m2}}(i))$$

$\mathrm{ret}_{\mathtt{m2}}$ is a function of the arguments of `m2`

We want to use it as a function of the arguments and local variables of `m1`

$\Rightarrow$ define parameter binding

[7]

Dynamic Memory Requirement Estimation

How much memory is needed to execute the following program?

```
void m0(int m) {
  for (c = 0; c < m; c++) {
    m1(c);                          /*S1*/
    B[] m2Arr = m2(2*m-c);/*S2*/
  }
}
void m1(int k) {
  for (i = 1; i <= k; i++) {
    A a = new A();                  /*S3*/
    B[] dummyArr = m2(i); /*S4*/
  }
}
B[] m2(int n) {
  B[] arrB = new B[n];              /*S5*/
  for (j = 1; j <= n; j++)
    B b = new B();                  /*S6*/
  return arrB;
}
```

$$
\begin{aligned}
I = \{ & \mathtt{m0}(m) \rightarrow \mathtt{S1}(c) : 0 \leq c < m; \\
& \mathtt{m0}(m) \rightarrow \mathtt{S2}(c) : 0 \leq c < m; \\
& \mathtt{m1}(k) \rightarrow \mathtt{S3}(i) : 1 \leq i \leq k; \\
& \mathtt{m1}(k) \rightarrow \mathtt{S4}(i) : 1 \leq i \leq k; \\
& \mathtt{m2}(n) \rightarrow \mathtt{S5}(); \\
& \mathtt{m2}(n) \rightarrow \mathtt{S6}(j) : 1 \leq j \leq n \}
\end{aligned}
$$

[7]

Dynamic Memory Requirement Estimation

How much memory is needed to execute the following program?

```
void m0(int m) {
  for (c = 0; c < m; c++) {
    m1(c);                          /*S1*/
    B[] m2Arr = m2(2*m-c);/*S2*/
  }
}
void m1(int k) {
  for (i = 1; i <= k; i++) {
    A a = new A();                  /*S3*/
    B[] dummyArr = m2(i);  /*S4*/
  }
}
B[] m2(int n) {
  B[] arrB = new B[n];              /*S5*/
  for (j = 1; j <= n; j++)
    B b = new B();                  /*S6*/
  return arrB;
}
```

$$I = \{ \mathtt{m0}(m) \rightarrow \mathtt{S1}(c) : 0 \leq c < m;$$
$$\mathtt{m0}(m) \rightarrow \mathtt{S2}(c) : 0 \leq c < m;$$
$$\mathtt{m1}(k) \rightarrow \mathtt{S3}(i) : 1 \leq i \leq k;$$
$$\mathtt{m1}(k) \rightarrow \mathtt{S4}(i) : 1 \leq i \leq k;$$
$$\mathtt{m2}(n) \rightarrow \mathtt{S5}();$$
$$\mathtt{m2}(n) \rightarrow \mathtt{S6}(j) : 1 \leq j \leq n \}$$

$$B^{\mathtt{m0}} = \{ (\mathtt{m0}(m) \rightarrow \mathtt{S1}(c)) \rightarrow \mathtt{m1}(c);$$
$$(\mathtt{m0}(m) \rightarrow \mathtt{S2}(c)) \rightarrow \mathtt{m2}(2m - c) \}$$
$$B^{\mathtt{m1}} = \{ (\mathtt{m1}(k) \rightarrow \mathtt{S4}(i)) \rightarrow \mathtt{m2}(i) \}$$

[7]

Dynamic Memory Requirement Estimation

```
void m1(int k) {
    for (i = 1; i <= k; i++) {
        A a = new A();                    /* S3 */
        B[] dummyArr = m2(i);             /* S4 */
    }
}
```

$$\mathrm{cap}_{\mathtt{m1}}(k) = \sum_{1 \le i \le k} (1 + \mathrm{ret}_{\mathtt{m2}}(i))$$

$\mathrm{ret}_{\mathtt{m2}}$ is a function of the arguments of `m2`

We want to use it as a function of the arguments and local variables of `m1`

$\Rightarrow$ define parameter binding

[7]

Dynamic Memory Requirement Estimation

```
void m1(int k) {
    for (i = 1; i <= k; i++) {
        A a = new A();                    /* S3 */
        B[] dummyArr = m2(i);             /* S4 */
    }
}
```

$$\mathrm{cap}_{\mathtt{m1}}(k) = \sum_{1 \le i \le k} (1 + \mathrm{ret}_{\mathtt{m2}}(i))$$

$$\mathrm{ret}_{\mathtt{m1}} = \{\, \mathtt{m1}(k) \to 0 \,\}$$

$$\mathrm{cap}_{\mathtt{m1}} = \left(\{\, (\mathtt{m1}(k) \to \mathtt{S3}(i)) \to 1 \,\} + \mathrm{ret}_{\mathtt{m2}} \circ B^{\mathtt{m1}}\right) \circ M$$

$$\mathrm{memRq}_{\mathtt{m1}} = \mathrm{cap}_{\mathtt{m1}} + \left(\mathrm{memRq}_{\mathtt{m2}} \circ B^{\mathtt{m1}} \circ M\right)$$

[7]

Dynamic Memory Requirement Estimation

```
void m1(int k) {
    for (i = 1; i <= k; i++) {
        A a = new A();                    /* S3 */
        B[] dummyArr = m2(i);             /* S4 */
    }
}
```

$$\mathrm{cap}_{\mathtt{m1}}(k) = \sum_{1 \le i \le k} (1 + \mathrm{ret}_{\mathtt{m2}}(i))$$

$$\mathrm{ret}_{\mathtt{m1}} = \{\, \mathtt{m1}(k) \to 0 \,\}$$

$$\mathrm{cap}_{\mathtt{m1}} = \left(\{\, (\mathtt{m1}(k) \to \mathtt{S3}(i)) \to 1 \,\} + \mathrm{ret}_{\mathtt{m2}} \circ B^{\mathtt{m1}}\right) \circ M$$

$$= \left\{ \mathtt{m1}(k) \to \frac{3}{2}k + \frac{1}{2}k^2 : k \ge 1 \right\}$$

$$\mathrm{memRq}_{\mathtt{m1}} = \mathrm{cap}_{\mathtt{m1}} + \left(\mathrm{memRq}_{\mathtt{m2}} \circ B^{\mathtt{m1}} \circ M\right)$$

$$= \left\{ \mathtt{m1}(k) \to \max\left(\frac{5}{2}k + \frac{1}{2}k^2\right) : k \ge 1 \right\}$$

[7]

Dynamic Memory Requirement Estimation

```
void m0(int m) {
    for (c = 0; c < m; c++) {
        m1(c);                              /* S1 */
        B[] m2Arr = m2(2 * m - c);          /* S2 */
    }
}
```

$$B^{\mathtt{m0}} = \{\, (\mathtt{m0}(k) \to \mathtt{S1}(c)) \to \mathtt{m1}(c); (\mathtt{m0}(k) \to \mathtt{S2}(c)) \to \mathtt{m2}(2m - c) \,\}$$

$$\mathrm{ret}_{\mathtt{m0}} = \{\, \mathtt{m0}(m) \to 0 \,\}$$

$$\mathrm{cap}_{\mathtt{m0}} = (\mathrm{ret}_{\mathtt{m1}} + \mathrm{ret}_{\mathtt{m2}}) \circ B^{\mathtt{m0}} \circ M$$

$$\mathrm{memRq}_{\mathtt{m0}} = \mathrm{cap}_{\mathtt{m0}} + \left((\mathrm{memRq}_{\mathtt{m1}} + \mathrm{memRq}_{\mathtt{m2}}) \circ B^{\mathtt{m0}} \circ M\right)$$

[7]

Dynamic Memory Requirement Estimation

```
void m0(int m) {
    for (c = 0; c < m; c++) {
        m1(c);                                  /* S1 */
        B[] m2Arr = m2(2 * m - c);              /* S2 */
    }
}
```

$$B^{\mathtt{m0}} = \{\, (\mathtt{m0}(k) \to \mathtt{S1}(c)) \to \mathtt{m1}(c); (\mathtt{m0}(k) \to \mathtt{S2}(c)) \to \mathtt{m2}(2m - c) \,\}$$

$$\mathrm{ret}_{\mathtt{m0}} = \{\, \mathtt{m0}(m) \to 0 \,\}$$

$$\mathrm{cap}_{\mathtt{m0}} = (\mathrm{ret}_{\mathtt{m1}} + \mathrm{ret}_{\mathtt{m2}}) \circ B^{\mathtt{m0}} \circ M$$

$$= \left\{ \mathtt{m0}(m) \to \frac{1}{2}m + \frac{3}{2}m^2 : m \geq 1 \right\}$$

$$\mathrm{memRq}_{\mathtt{m0}} = \mathrm{cap}_{\mathtt{m0}} + ((\mathrm{memRq}_{\mathtt{m1}} + \mathrm{memRq}_{\mathtt{m2}}) \circ B^{\mathtt{m0}} \circ M)$$

$$= \left\{ \mathtt{m0}(m) \to \max\left(-2 + 2m + 2m^2, \frac{5}{2}m + \frac{3}{2}m^2\right) : m \geq 1 \right\}$$

Outline

Positive Powers

Definition (Power of a Relation)

Let R be a Presburger relation and k a positive integer, then power k of relation R is defined as

$$R^k := \begin{cases} R & \text{if } k = 1 \\ R \circ R^{k-1} & \text{if } k \geq 2. \end{cases}$$

Positive Powers

Definition (Power of a Relation)

Let R be a Presburger relation and k a positive integer, then power k of relation R is defined as

$$R^k := \begin{cases} R & \text{if } k = 1 \\ R \circ R^{k-1} & \text{if } k \geq 2. \end{cases}$$

Example

$$R = \{ (x) \rightarrow (x+1) \}$$
$$R^k = \{ (x) \rightarrow (x+k) : k \geq 1 \}$$

Transitive Closures

Definition (Transitive Closure of a Relation)

Let R be a Presburger relation, then the transitive closure R^+ of R is the union of all positive powers of R,

$$R^+ := \bigcup_{k \geq 1} R^k.$$

Transitive Closures

Definition (Transitive Closure of a Relation)

Let R be a Presburger relation, then the transitive closure R^+ of R is the union of all positive powers of R,

$$R^+ := \bigcup_{k \geq 1} R^k.$$

Example

$$R = \{ (x) \to (x+1) \}$$
$$R^k = \{ (x) \to (x+k) : k \geq 1 \}$$
$$R^+ = \{ (x) \to (y) : \exists k \geq 1 : y = x + k \} = \{ (x) \to (y) : y \geq x + 1 \}$$

Transitive Closures

Definition (Transitive Closure of a Relation)

Let R be a Presburger relation, then the transitive closure R^+ of R is the union of all positive powers of R,

$$R^+ := \bigcup_{k \geq 1} R^k.$$

Example

$$R = \{ (x) \to (x+1) \}$$
$$R^k = \{ (x) \to (x+k) : k \geq 1 \}$$
$$R^+ = \{ (x) \to (y) : \exists k \geq 1 : y = x + k \} = \{ (x) \to (y) : y \geq x + 1 \}$$

Definition (Transitive Closure of a Relation, Alternative)

Inductive definition:

$$R^+ := R \cup \left(R \circ R^+ \right)$$

Transitive Closures — Approximation

Fact

Given a Presburger relation R, the power R^k (with k a parameter) and the transitive closure R^+ may not be Presburger relations.

Example

$$R = \{ (x) \to (2\,x) \}$$
$$R^k = \{ (x) \to (2^k\,x) \}$$

Transitive Closures — Approximation

Fact

Given a Presburger relation R, the power R^k (with k a parameter) and the transitive closure R^+ may not be Presburger relations.

Example

$$R = \{ (x) \to (2\,x) \}$$
$$R^k = \{ (x) \to (2^k\,x) \}$$

$\Rightarrow$ need for approximation

- overapproximation $R^{\overline{+}}$
- underapproximation $R^{\underline{+}}$

[15, 22]

Transitive Closures — Approximation

Fact

Given a Presburger relation R, the power R^k (with k a parameter) and the transitive closure R^+ may not be Presburger relations.

Example

$$R = \{ (x) \rightarrow (2\,x) \}$$
$$R^k = \{ (x) \rightarrow (2^k\,x) \}$$

$\Rightarrow$ need for approximation

- overapproximation $R^{\overline{+}}$
- underapproximation $R^{\underline{+}}$

Note

Do not use transitive closures if there is an alternative.

Part II

Tools

Outline

Availability — Representation

[20]

$$\{ A(i) : 0 \leq i \leq n; B(i,j) : \exists \alpha : i = 2\alpha \}$$

- Named (and nested) spaces: `isl`

```
[n] -> { A[i]: 0 <= i <= n; B[i,j]: exists a: i = 2 a }
```

In `omega(+)`:

Availability — Representation

[20]

$$\{ A(i) : 0 \leq i \leq n; B(i,j) : \exists \alpha : i = 2\alpha \}$$

- Named (and nested) spaces: `isl`

```
[n] -> { A[i]: 0 <= i <= n; B[i,j]: exists a: i = 2 a }
```

In omega(+):

A

padding

B

```
symbolic n;
{ [0, i, 0]: 0 <= i <= n } union { [1, i,j]: exists a: i = 2 a }
```

Availability — Representation

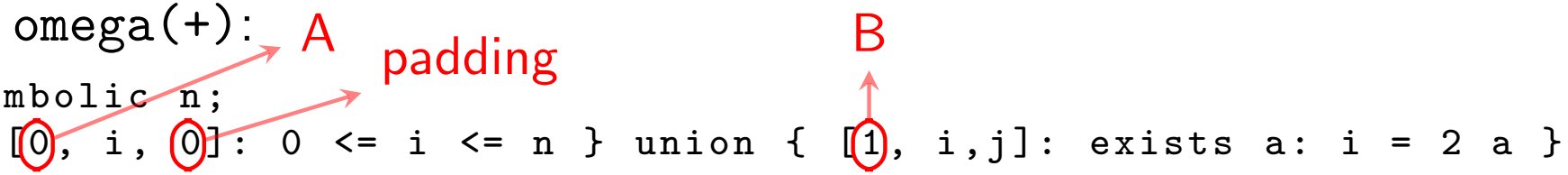
[20]

$$\{ A(i) : 0 \leq i \leq n; B(i,j) : \exists \alpha : i = 2\alpha \}$$

- Named (and nested) spaces: `isl`

```
[n] -> { A[i]: 0 <= i <= n; B[i,j]: exists a: i = 2 a }
```

In `omega(+)`:



- Presburger sets and relations: `isl`, `omega(+)`

Availability — Representation

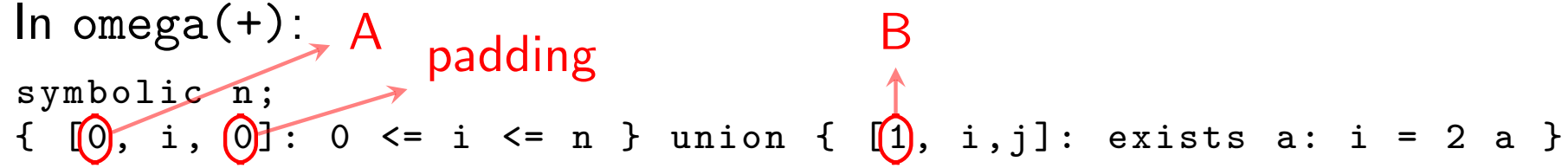
[20]

$$\{ A(i) : 0 \leq i \leq n; B(i,j) : \exists \alpha : i = 2\alpha \}$$

- Named (and nested) spaces: `isl`

```
[n] -> { A[i]: 0 <= i <= n; B[i,j]: exists a: i = 2 a }
```

In `omega(+)`:



- Presburger sets and relations: `isl`, `omega(+)`

In `PolyLib`:

Moreover: `PolyLib` deals with rational sets (polyhedra)

Availability — Representation (2)

- Uninterpreted functions: `omega(+)`
  - $\Rightarrow$ arity can be greater than 0
  - $\Rightarrow$ not available in `isl` (yet)

  Note: support in `omega(+)` for uninterpreted functions is very restrictive
  - arguments need to be prefix of input/output dimensions
    - $\Rightarrow$ essentially symbolic constants

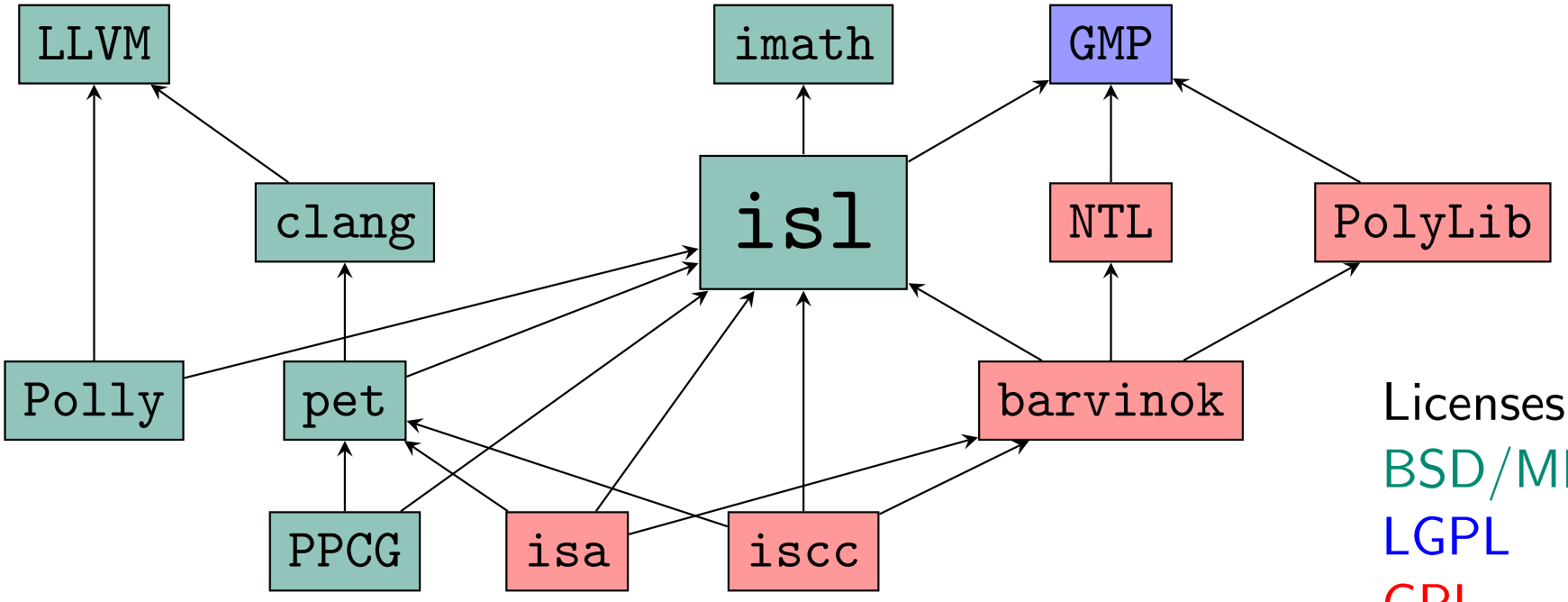
Availability — Operations

	union	intersection	difference	emptiness	domain range inverse application
`PolyLib`	✓	✓	in $\mathbb{Z}$		
`PPL`	✓	✓		✓	
`PIP`				lexmin	
`omega(+)`	✓	✓	✓	✓	✓
`isl`	✓	✓	✓	✓	✓
`barvinok`				card	
`bernstein`					
`LattE`				card	

Availability — Operations (2)

	lexmin	cardinality	bounds on polynomials	weighted counting	transitive closure
`PolyLib`		✓			
`PPL`	✓				
`PIP`	✓				
`omega(+)`	Presburger				"$R^{\pm}$"
`isl`	✓		✓		R^{+}
`barvinok`	partial	✓		✓	
`bernstein`			✓		
`LattE`		✓		✓	

isl and Related Libraries and Tools



isl: manipulates parametric affine sets and relations

barvinok: counts elements in parametric affine sets and relations

pet: extracts polyhedral model from clang AST

PPCG: Polyhedral Parallel Code Generator

iscc: interactive calculator

isa: prototype tool set including derivation of process networks and equivalence checker

Overview of `isl`

`isl` is a thread-safe C library for manipulating **integer sets and relations**

- bounded by *affine constraints*
- involving *symbolic constants* and
- *existentially quantified variables*

and **quasi-affine** and **quasi-polynomial functions** on such domains

Supported operations by core library include

- *intersection*
- *union*
- *set difference*
- *integer projection*
- *coalescing*
- *closed convex hull*
- *sampling*, *scanning*
- *integer affine hull*
- *lexicographic optimization*
- *transitive closure* (approx.)
- *parametric vertex enumeration*
- *bounds on quasipolynomials*

Polyhedral compilation library

- *schedule trees*
- *dataflow analysis*
- *scheduling*
- *AST generation*

PPCG

PPCG (http://ppcg.gforge.inria.fr/)

- Input: C code
- Output: CUDA or OpenCL code for GPGPUs

PPCG

PPCG (http://ppcg.gforge.inria.fr/)

- Input: C code
- Output: CUDA or OpenCL code for GPGPUs

Steps:

- extract polyhedral model from C code (pet)
- dependence analysis (isl)
- scheduling
  - expose parallelism and tiling opportunities (isl)
  - perform tiling (isl)
  - separate into parts mapped to host, GPU blocks and GPU threads
- memory management
  - add transfers of data to/from GPU
  - detect array reference groups
  - allocate groups to registers and shared memory
- generate AST (isl)

PPCG Example — Input

```
void matmul(int M, int N, int K,
            float A[static const restrict M][K],
            float B[static const restrict K][N],
            float C[static const restrict M][N])
{
  for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) {
S1:   C[i][j] = 0;
      for (int k = 0; k < K; k++)
S2:     C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
}
```

Options:

```
--ctx="[M,N,K] -> { : M = N = K = 256 }"
--sizes="{ kernel[i] -> tile[16,16,16];
           kernel[i] -> block[8,16] }"
--pet-autodetect
```

PPCG Example — Output

```
long b0 = blockIdx.y, b1 = blockIdx.x;
long t0 = threadIdx.y, t1 = threadIdx.x;
__shared__ float s_A[16][16];
float p_C[2][1];
__shared__ float s_B[16][16];

for (long g9 = 0; g9 <= 15; g9 += 1) {
  for (long c0 = t0; c0 <= 15; c0 += 8)
    s_B[c0][t1] = B[(16 * g9 + c0) * (256) + 16 * b1 + t1];
  for (long c0 = t0; c0 <= 15; c0 += 8)
    s_A[c0][t1] = A[(16 * b0 + c0) * (256) + t1 + 16 * g9];
  __syncthreads();
  if (g9 == 0) {
    p_C[0][0] = (0);
    p_C[1][0] = (0);
  }
  for (long c3 = 0; c3 <= 15; c3 += 1) {
    p_C[0][0] = (p_C[0][0] + (s_A[t0][c3] * s_B[c3][t1]));
    p_C[1][0] = (p_C[1][0] + (s_A[t0 + 8][c3] * s_B[c3][t1]));
  }
  __syncthreads();
}
C[(16 * b0 + t0) * (256) + 16 * b1 + t1] = p_C[0][0];
C[(16 * b0 + t0 + 8) * (256) + 16 * b1 + t1] = p_C[1][0];
```

CARP Project

Design tools and techniques to aid **Correct and Efficient Accelerator Programming**

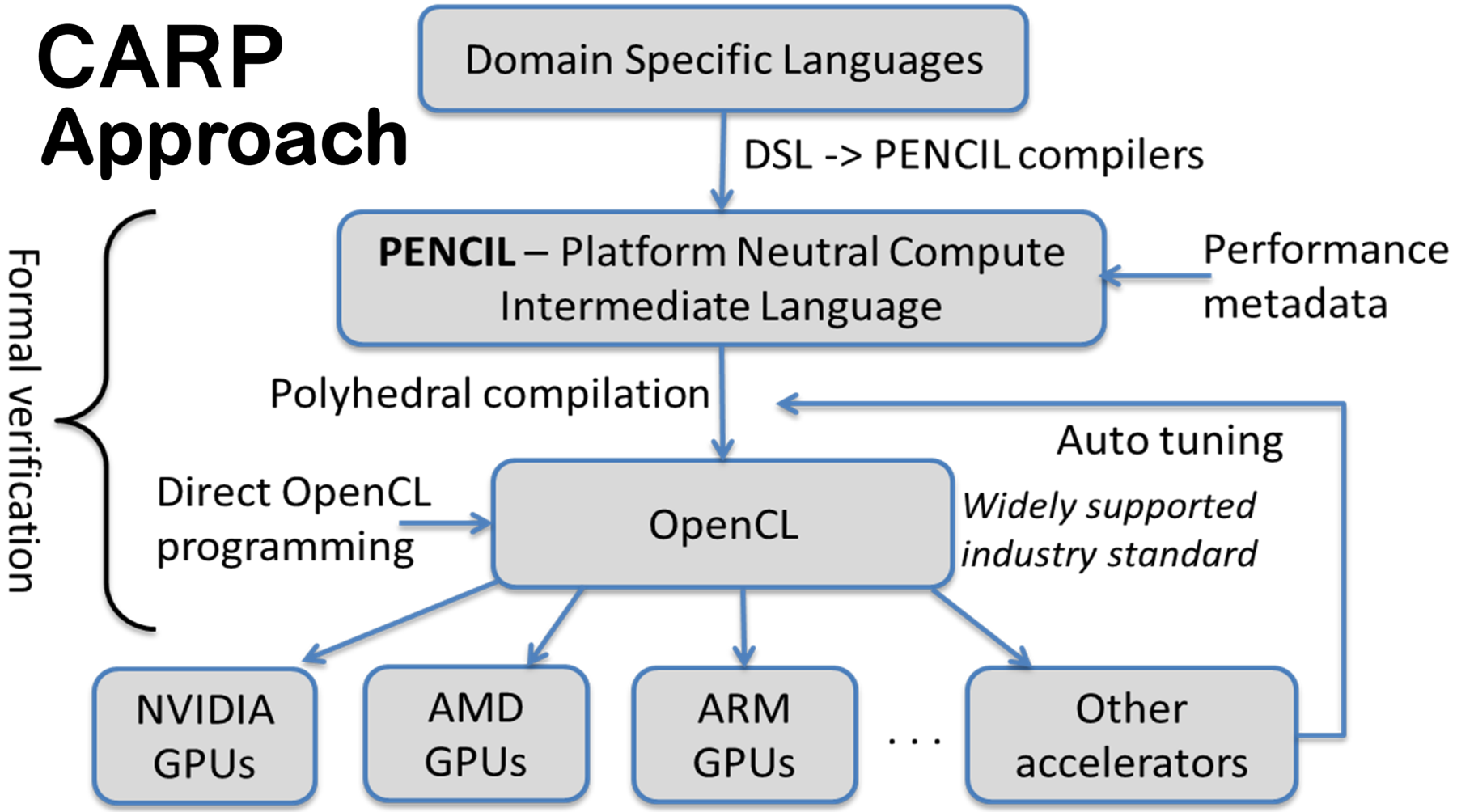
Key areas:

- High level programming models
- Advanced compilation techniques
- Formal verification

Partners:

- Imperial College London (UK)
- ENS (FR)
- ARM (UK)
- Realeyes (ES)
- RWTH Aachen University (DE)
- Monoidics (UK)
- University of Twente (NL)
- Rightware (FI)

CARP Approach



References I

[1] R. Baghdadi, A. Cohen, S. Verdoolaege, and K. Trifunovic.
Improved loop tiling based on the removal of spurious false dependences.
TACO, 9(4):52, 2013.

[2] R. Bagnara, P. M. Hill, and E. Zaffanella.
The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems.
Science of Computer Programming, 72(1–2):3–21, 2008.

[3] D. Barthou, A. Cohen, and J.-F. Collard.
Maximal static expansion.
In *POPL '98: Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 98–106, New York, NY, USA, 1998. ACM.

References II

[4] D. Barthou, J.-F. Collard, and P. Feautrier.
Fuzzy array dataflow analysis.
J. Parallel Distrib. Comput., 40(2):210–226, 1997.

[5] C. Bastoul.
Code generation in the polyhedral model is easier than you think.
In *PACT '04: Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, pages 7–16, Washington, DC, USA, 2004. IEEE Computer Society.

[6] C. Chen.
Polyhedra scanning revisited.
SIGPLAN Not., 47(6):499–508, June 2012.

References III

[7] P. Clauss, F. J. Fernandez, D. Garbervetsky, and S. Verdoolaege.
Symbolic polynomial maximization over convex sets and its application to memory requirement estimation.
IEEE Transactions on VLSI Systems, 17(8):983–996, Aug. 2009.

[8] P. Cousot and N. Halbwachs.
Automatic discovery of linear restraints among variables of a program.
In *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages*, pages 84–96, Tucson, Arizona, 1978. ACM Press.

[9] P. Feautrier.
Dataflow analysis of array and scalar references.
International Journal of Parallel Programming, 20(1):23–53, 1991.

References IV

[10] P. Feautrier and C. Lengauer.
The polyhedron model.
In D. Padua, editor, *Encyclopedia of Parallel Computing*, pages 1581–1592. Springer, 2011.

[11] S. Girbal, N. Vasilache, C. Bastoul, A. Cohen, D. Parello, M. Sigler, and O. Temam.
Semi-automatic composition of loop transformations for deep parallelism and memory hierarchies.
Int. J. Parallel Program., 34(3):261–317, June 2006.

[12] T. Grosser, A. Groesslinger, and C. Lengauer.
Polly - performing polyhedral optimizations on a low-level intermediate representation.
Parallel Processing Letters, 22(04), 2012.

References V

[13] W. Kelly.
Optimization within a unified transformation framework.
Technical Report CS-TR-3725, Dept. of CS, Univ. of Maryland, College Park, 1996.

[14] W. Kelly, V. Maslov, W. Pugh, E. Rosser, T. Shpeisman, and D. Wonnacott.
The Omega library.
Technical report, University of Maryland, Nov. 1996.

[15] W. Kelly, W. Pugh, E. Rosser, and T. Shpeisman.
Transitive closure of infinite graphs and its applications.
Int. J. Parallel Program., 24(6):579–598, 1996.

References VI

[16] V. Loechner and D. K. Wilde.
Parameterized polyhedra and their vertices.
International Journal of Parallel Programming, 25(6):525–549, Dec. 1997.

[17] V. Maslov.
Lazy array data-flow dependence analysis.
In H.-J. Boehm, B. Lang, and D. M. Yellin, editors, *POPL*, pages 311–325. ACM Press, 1994.

[18] S. Verdoolaege.
isl: An integer set library for the polyhedral model.
In K. Fukuda, J. Hoeven, M. Joswig, and N. Takayama, editors, *Mathematical Software - ICMS 2010*, volume 6327 of *Lecture Notes in Computer Science*, pages 299–302. Springer, 2010.

References VII

[19] S. Verdoolaege.
Polyhedral process networks.
Springer, 2010.

[20] S. Verdoolaege.
Counting affine calculator and applications.
In *First International Workshop on Polyhedral Compilation Techniques (IMPACT'11)*, Chamonix, France, Apr. 2011.

[21] S. Verdoolaege, J. Carlos Juega, A. Cohen, J. Ignacio Gómez, C. Tenllado, and F. Catthoor.
Polyhedral parallel code generation for CUDA.
ACM TACO, 9(4):54, 2013.

References VIII

[22] S. Verdoolaege, A. Cohen, and A. Beletska.
Transitive closures of affine integer tuple relations and their overapproximations.
In *Proceedings of the 18th International Conference on Static Analysis*, SAS'11, pages 216–232, Berlin, Heidelberg, 2011. Springer-Verlag.

[23] S. Verdoolaege and T. Grosser.
Polyhedral extraction tool.
In *Second International Workshop on Polyhedral Compilation Techniques (IMPACT'12)*, Paris, France, Jan. 2012.

[24] S. Verdoolaege, S. Guelton, T. Grosser, and A. Cohen.
Schedule trees.
In *Proceedings of the 4th International Workshop on Polyhedral Compilation Techniques*, Vienna, Austria, Jan. 2014.

References IX

[25] S. Verdoolaege, H. Nikolov, and T. Stefanov.
On demand parametric array dataflow analysis.
In *Third International Workshop on Polyhedral Compilation Techniques (IMPACT'12)*, Berlin, Germany, Jan. 2013.

[26] S. Verdoolaege, R. Seghir, K. Beyls, V. Loechner, and M. Bruynooghe.
Counting integer points in parametric polytopes using Barvinok's rational functions.
Algorithmica, 48(1):37–66, June 2007.

[27] D. K. Wilde.
A library for doing polyhedral operations.
Technical Report 785, IRISA, Rennes, France, 1993.